METHODS, SYSTEMS, AND SOFTWARE FOR PREDICTING BIOCHEMICAL NETWORKS OR PATHWAYS

COPYRIGHT NOTIFICATION

CROSS-REFERENCES TO RELATED APPLICATIONS

[0002] This application claims the benefit of U.S. Provisional Application No. 60/489,317, filed July 22, 2003, the disclosure of which is incorporated by reference in its entirety for all purposes.

STATEMENT AS TO RIGHTS TO INVENTIONS MADE UNDER FEDERALLY SPONSORED RESEARCH AND DEVELOPMENT

[0003] This invention was made with government support under Grant No. BES-9911447 awarded by the National Science Foundation, Grant No. DE-FG03-01ER63111 awarded by the Department of Energy, and Grant No. N00014-00-1-0749 awarded by the Office of Naval Research. The government may have certain rights in the invention.

FIELD OF THE INVENTION

[0004] The present invention relates generally to predicting or inferring biochemical networks or pathways. In particular, the invention provides methods, systems, and computer program products for automatic biochemical network or pathway inference.

BACKGROUND OF THE INVENTION

[0005] Automated methods for biochemical pathway inference or prediction are becoming increasingly important for understanding biological processes in living and synthetic systems. With the availability of data on complete genomes and increasing information about enzyme-catalyzed biochemistry it is becoming feasible to

approach this problem computationally. However, even with the availability of a genomic blueprint for a living system and functional annotations for its putative genes, the experimental elucidation of its biochemical processes is typically still a daunting task. Though it is possible to organize genes by broad functional roles, piecing them together manually into consistent biochemical pathways can quickly become intractable.

[0006] A number of metabolic pathway reconstruction tools have been alleged since the availability of the first microbial genome, *H. influenza* (Fleischmann et al. (1995) "Whole-genome random sequencing and assembly of Haemophilus influenzae Rd," Science 269:469-512). These include PathoLogic (Karp & Riley, Representations of metabolic knowledge: Pathways. In Second International Conference on Intelligent Systems for Molecular Biology (Altman, R., Brutlag, D., Karp, P., Lathrop, R. & Searls, D., eds), AAAI Press (1994)), MAGPIE (Gaasterland & Selkov (1995) "Automatic Reconstruction of Metabolic Networks Using Incomplete Information," Intelligent Systems for Molecular Biology 3:127-135 and Gaasterland & Sensen (1996) "MAGPIE: automated genome interpretation," Trends Genet 12(2):76-78), WIT (Overbeek et al. (2000) "Wit: integrated system for high-throughput genome sequence analysis and metabolic reconstruction," Nucleic Acids Res 28(1):123-125) and PathFinder (Goesmarm et al. (2002) "PathFinder: reconstruction and dynamic visualization of metabolic pathways," Bioinformatics 18(1):124-129, which are each incorporated by reference). The goal of most pathway inference methods has generally been to match putatively identified enzymes with known, or "reference", pathways. Although reconstruction can be a useful starting point for elucidating the metabolic capabilities of an organism based upon prior pathway knowledge, reconstructed pathways often have many missing enzymes, even in essential pathways.

[0007] In addition, the issue of redefining microbial biochemical pathways based on "missing" enzymes is often of consequence since there are many examples of alternatives to standard pathways in a variety of organisms (Cordwell (1999) "Microbial genomes and missing enzymes: redefining biochemical pathways," Arch Microbiol 172(5):269-279, which is incorporated by reference). Moreover, engineering a new pathway into an organism through, e.g., heterologous enzymes also typically requires the ability to infer new biochemical routes.

SUMMARY OF THE INVENTION

[0008] The present invention relates to the prediction of biochemical networks, routes, or pathways, including the inference of xenobiotic metabolism. To illustrate, the methods described herein can be utilized to search for biochemical networks or pathways in the context of a given taxonomical selection (e.g., a given genus, species, etc.), or in metabolic engineering procedures to identify or design biochemical networks or pathways among many other applications. In certain embodiments, for example, the computational approaches described herein abstract biochemical processes in terms of state-space in which compounds define the states and transformations between compounds define state-transitions. In these embodiments, biochemical network or pathway prediction typically includes searching this state-space, e.g., using an informed search technique. In some embodiments, symbolic computational approaches are used that include inferring biotransformational rules from molecular graphs of compounds in biocatalyst-catalyzed reactions. The biotransformational rules are then typically recursively applied to different compounds to generate novel metabolic networks that include new biotransformations and metabolites. In addition to methods of predicting biochemical networks or pathways, the invention also provides related computer program products and systems.

[0009] In one aspect, the invention relates to a method of predicting (e.g., searching, discovering, etc.) a biochemical network or pathway (e.g., a metabolic network or pathway, such as an anabolic or catabolic network or pathway). The method includes providing a population of compounds. The population comprises one or more input compounds and one or more output compounds. The method also includes defining at least one state-space that comprises the population of compounds. In addition, the method also includes identifying one or more candidate biochemical networks or pathways between at least one of the input compounds and at least one of the output compounds using at least one informed search technique (e.g., a greedy search, a uniform cost search, an A* search, etc.) to search the state-space. In certain embodiments, the informed search technique comprises chemical distances between compounds in the population of compounds as an admissible heuristic. Optionally, the method includes maintaining the admissibility of the heuristic by incorporating one or more penalties into an edge-cost associated with an edge in the state-space.

[0010] In some embodiments, the defining step comprises deriving one or more symbolic chemical substructural rules from biotransformations between two or more compounds in the population of compounds. In these embodiments, the identifying step typically comprises recursively applying the symbolic chemical substructural rules to a selected compound to predict the biochemical network or pathway. To further illustrate, the state-space is defined as two or more compound states and at least one state-transition between at least two of the compound states in certain embodiments. In these embodiments, at least one biocatalyst, at least one structural changes, at least one energetic change and/or the like generally describes the state-transition. Typically, at least one of the input and/or output compounds are described by one or more chemical descriptors. For example, at least one of the chemical descriptors is optionally selected from the list of descriptors provided in Figure 2. To further illustrate, one or more of the input and/or output compounds is defined by at least one state vector in some embodiments.

[0011] In another aspect, the invention provides a computer program product comprising a computer readable medium having one or more logic instructions for receiving data that defines at least one state-space comprising a population of compounds, which population comprises one or more input compounds and one or more output compounds. The computer readable medium also includes one or more logic instructions for identifying one or more candidate biochemical networks or pathways between at least one of the input compounds and at least one of the output compounds using at least one informed search technique to search the state-space. The computer readable medium generally comprises one or more of, e.g., a CD-ROM, a floppy disk, a tape, a flash memory device or component, a system memory device or component, a hard drive, a data signal embodied in a carrier wave, or the like. Exemplary source code is provided in the appendix.

[0012] In still another aspect, the invention relates to a system for predicting a biochemical network or pathway. The system comprises at least one computer having system software comprising one or more logic instructions for receiving data that defines at least one state-space comprising a population of compounds, which population comprises one or more input compounds and one or more output compounds. The system software also comprises one or more logic instructions for identifying one or more candidate biochemical networks or pathways

between at least one of the input compounds and at least one of the output compounds using at least one informed search technique to search the state-space.

[0013] The computer program products and systems of the invention include various embodiments. In certain embodiments, for example, the computer readable medium or system software includes at least one logic instruction for: deriving one or more symbolic chemical substructural rules from biotransformations between two or more compounds in the population of compounds, and recursively applying the symbolic chemical substructural rules to a selected compound to identify the candidate biochemical networks or pathways. To further illustrate, the received data optionally comprises compound states and state-transitions between the compound states in certain embodiments.

APPENDIX

[0014] This application is being filed with a paper appendix totaling 27 pages. This appendix provides example source code illustrating specific implementations of specific embodiments of the invention and is incorporated by reference.

BRIEF DESCRIPTION OF THE DRAWINGS

[0015] Figure 1 is a flow chart illustrating a method of predicting biochemical networks or pathways according to specific embodiments of the invention.

[0016] Figure 2 provides an alphabetical list of 145 descriptors used to represent chemical state-space according to one embodiment of the invention. As shown, atoms are represented by their International Union of Pure and Applied Chemistry (IUPAC) symbols. Single, double, and triple bonds are represented as the symbols -, =, and #, respectively.

[0017] Figure 3 schematically depicts certain known chemical successors of α-D-glucose (adg), denoted T^{adg}. More specifically, the schematically illustrated structure of adg is shown on the left side of the figure, while the structures of the successors are schematically shown on the right side of the figure.

[0018] Figure 4 schematically shows a best-first search algorithm to find network or pathway, $P^{0,L}$, from an input compound, x^0, to output compound x^L, using a heuristic evaluation function, F, according to one embodiment of the invention.

[0019] Figure 5 A and B illustrate example interfaces for predicting biochemical networks or pathways using a computer interface, possibly over a web page, according to specific embodiments of the present invention.

[0020] Figure 6 is a block diagram showing a representative example logic device in which various aspects of the present invention may be embodied.

[0021] Figure 7 is a block diagram illustrating an integrated system according to specific embodiments of the present invention.

[0022] Figure 8 provides tables showing the performance of the different search algorithms in state-space for four sample queries. Each table summarizes the results for a query from a starting compound to a goal compound. In each table, the columns contain the statistics for the computed pathways and the rows compare the results for the different search algorithms. In each table, from left to right the columns contain the algorithm (Alg); the total number of states explored in the search (M); the length of the path (L); the path cost (F), which is calculated using equation 8; the effective branching factor (b*), which is computed using equation 9; and the time required for the computation (t). For each table, the second row shows the results for breadth first search (BFS); the third row for depth first search (DFS); and the fourth row for A* search (A*).

[0023] Figure 9 schematically shows the visualization of a linear network or pathway according to one embodiment of the invention.

[0024] Figure 10 schematically depicts an exemplary vanillin synthetic pathway.

[0025] Figure 11 schematically shows a search algorithm according to one embodiment of the invention.

[0026] Figure 12 is a graph that schematically depicts combinatorial complexity. More specifically, this figure illustrates the expanded states, the pathways, and the worst case breadth-first search time as a function of pathway length. While the number of states increases linearly with path length, the number of pathways, and hence the search time, increases exponentially. The logarithmic regressions are shown where the abscissa is the path length.

[0027] Figure 13 shows the net equation, including all side compounds, for an optimal pathway for vanillin synthesis.

[0028] Figure 14 schematically shows multiple pathways from alpha-D-glucose to vanillin.

[0029] Figure 15 schematically shows the degradation of lignin to vanillin.

[0030] Figure 16 schematically illustrates an alcohol dehydrogenase (EC 1.1.99.9) transformation from abstract primary alcohol to abstract aldehyde showing the computed Δ^- and Δ^+ moieties. The Δ^- moiety is the subgraph that is in X_s but not in X_p. The Δ^+ moiety is the subgraph that is in X_p but not in X_s.

[0031] Figure 17 schematically illustrates CYP2D6 (EC 1.14.14.1) reactions in the Kyoto Encyclopedia of Genes and Genomes (KEGG). Compounds are either abstract (i.e., contain one or more Markush "R" groups) or normal (i.e., have unique structures).

[0032] Figure 18 schematically depicts the application of an alcohol dehydrogenase rule to ethanol.

[0033] Figure 19 schematically shows a rule application algorithm according to one embodiment of the invention.

[0034] Figure 20 schematically depicts the de novo prediction of ethanol metabolism. Ethanol is in the center of the figure. The highlighted transformations are the activation of the alcohol to an aldehyde by alcohol dehydrogenase (EC 1.1.99.20), then to an acid by aldehyde oxidase (EC 1.2.3.1), respectively. Not shown, but in the next iteration is the O-glycosylation of the aldehyde by beta-Glucuronidase (EC 3.2.1.31).

[0035] Figure 21 schematically shows the de novo prediction of furfuryl metabolism. Furfurol is in the center of the figure. The highlighted transformation between compound furfurol and compound NO0482 (up and to the left) is an O-glycosylation by beta-Glucuronidase (EC 3.2.1.31). The highlighted transformations below furfurol are the activation of the alcohol to an aldehyde (N00479, furfural) by alcohol dehydrogenase (EC 1.1.99.20), then to an acid by aldehyde oxidase (EC 1.2.3.1), respectively. The acid is identified by the algorithm as being in the KEGG database (by graph similarity) as 2-Furoate (C01546). In the next iteration, not shown, the acid is finally O-glycosylated by beta-Glucuronidase (EC 3.2.1.31).

DETAILED DISCUSSION OF THE INVENTION

I. DEFINITIONS

[0036] Before describing the present invention in detail, it is to be understood that this invention is not limited to particular methods, systems, computers, or computer readable media, which can, of course, vary. It is also to be understood that the terminology used herein is for the purpose of describing particular embodiments only, and is not intended to be limiting. Further, unless defined otherwise, all technical and scientific terms used herein have the same meaning as commonly understood by one of ordinary skill in the art to which this invention pertains. In describing and claiming the present invention, the following terminology and grammatical variants will be used in accordance with the definitions set forth below.

[0037] A "**biocatalyst**" refers to a catalyst that reduces the activation energy of a biochemical reaction involving input and output compounds. Exemplary biocatalysts include protein-based catalysts (e.g., "enzymes"), DNA-based catalysts (e.g., "DNAzymes"), RNA-based catalysts (e.g., "ribozymes"), etc. To further illustrate, a biocatalyst can be associated with one or more state-transitions.

[0038] A "**biochemical network**" refers to a set of two or more biochemical pathways. In some embodiments, at least two of the biochemical pathways in a given biochemical network are interconnected or interrelated with one another.

[0039] A "**biochemical pathway**" refers to a biochemical reaction or sequence of biochemical reactions that begins with one or more input compounds and yields one or more output compounds. A biochemical pathway may be endogenous or xenobiotic to a particular organism. One or more reaction steps in a biochemical pathway are typically catalyzed by one or more biocatalysts. To further illustrate, biochemical pathways include the metabolism of drugs, environmental toxicants, food additives, etc. in essentially any selected organism or group of organisms, including engineered or model organisms.

[0040] A "**descriptor**" refers to something that serves to describe or identify an item. For example, chemical descriptors can be used to describe a compound in terms of, e.g., the number and/or types of constituent atoms of the compound, the number and/or types bonds of the compound, and/or other attributes of the compound.

[0041] A "**heuristic search technique**" or "**informed search technique**" refers to an exploratory problem-solving procedure that utilizes self-educating techniques (as the evaluation of feedback) to improve performance. Exemplary informed search techniques include, e.g., a greedy search, a uniform cost search, an A* search, or the like.

[0042] An "**input compound**" or "initial compound" refers to a reactant or a representation of a reactant in a given chemical reaction. An "output compound," "destination compound," or "successor compound" refers to a product or a representation of a product in a given chemical reaction.

[0043] A "**population**" refers to a collection of at least two molecule or compound types, e.g., 2, 3, 4, 5, 10, 20, 50, 100, 1,000 or more molecule or compound types.

[0044] The term "**state-space**" refers a population of states (e.g., chemical compounds, etc.) and to transitions (e.g., chemical reactions, etc.) between those states in the population.

[0045] The term "**state-space transition**" or "**state-transition**" refers to a known or novel biochemical transformation that can be associated with biocatalysts and that is associated with biochemical substructural rules.

II. BIOCHEMICAL NETWORK OR PATHWAY PREDICTION

[0046] The methods of the invention predict biochemical routes by reasoning over transformations using chemical and biological information. More specifically, the present invention provides computational approaches for automated network or pathway prediction that are useful for exploring plausible biochemical routes underlying various biological processes. Although essentially any programming language is optionally utilized to implement the methods described herein, certain specific embodiments referred to herein are implemented in Common Lisp (*see, e.g.,* Graham, ANSI Common LISP, 1st Ed., Prentice Hall (1995) and Norvig, Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp, Morgan Kaufmann (1991), which is incorporated by reference. In addition, a flexible web-based interactive system, called PathMiner, that embodies aspects of the present invention is described further in, e.g., McShan et al. (2003) "PathMiner: Predicting Metabolic Pathways by Heuristic Search," Bioinformatics 19(13):1692-1698, which is incorporated by reference. Aspects of this system are also described in, e.g., McShan et

al. (2004) "Symbolic inference of xenobiotic metabolism" Pac Symp Biocomput. 545-56 and McShan et al. (2004) "Heuristic search for metabolic engineering: *de novo* synthesis of vanillin" J. Comp. & Chem. Eng. (in press), which are both incorporated by reference. There are at least two broad biological applications of the invention. First, to investigate networks or pathways in an organism using information about its functionally characterized proteins or other biocatalysts. Second, to synthesize novel networks or pathways for engineering new biochemical capabilities. These and other features of the invention will be apparent upon complete review of this disclosure, including the examples provided below.

[0047] Going beyond standard networks or pathways is one objective of the present invention, which uses chemically motivated heuristics to guide the search for networks or pathways in certain embodiments. As such, it complements pre-existing approaches like PathoLogic (referred to above), which find the best candidate reference pathways and the corresponding genes in a living system. Other approaches to pathway synthesis have also included the work of Sesseriotis and Bailey (Seressiotis & Bailey (1988) "Mps: an artificially intelligent software system for the analysis and synthesis of metabolic pathways," Biotechnology and Bioengineering 31:587-602, which is incorporated by reference), and later, Mavrovouniotis's approach for pathway generation based on the consideration of thermodynamic feasibility of reactions (Mavrovouniotis, "Identification of Qualitatively Feasible Metabolic Pathways" In: Artificial Intelligence and Molecular Biology, (Hunter (Ed.)) AAAI (1993), which is incorporated by reference). Further, the present invention can be used interactively to search for biochemical routes in the context of specific organisms or to identify synthetic pathways.

[0048] In overview, the present invention abstracts biochemical processes in terms of a biochemical state-space: compounds define the states and transformations between compounds define the state-transitions. Network or pathway prediction is then considered as a problem of searching the biochemical state-space. State-space and an embodiment of an algorithm for predicting networks or pathways through search are described below. To further illustrate, Figure 1 provides a flow chart illustrating an example method according to specific embodiments of the invention. As shown, the method includes as follows: providing a population of compounds having input and output compounds (A1); defining a state-space that

comprises the population of compounds (A2); and identifying a candidate biochemical networks or pathway between an input compound and an output compound using a heuristic search technique to search the state-space (A3).

A. THE BIOCHEMICAL STATE-SPACE

[0049] The notion of the biochemical state-space is based on resolving biocatalyst biochemistry into two components. The first component is the chemical component, which represents transformations between, e.g., metabolites. The second component is the biocatalytic component, which involves transformations catalyzed by enzymes or other biocatalysts. This logical separation between biocatalysts and the chemistry they catalyze is evolutionarily plausible and functionally relevant, since a biocatalyst can often catalyze multiple transformations. Additional details relating to abstracting the interaction of a biocatalyst with a chemical transformation are provided in, e.g., Karp (Karp & Riley (1994), *supra*). By considering chemical transformations and biocatalysts separately, they can be dealt with rationally to infer plausible networks or pathways.

1. COMPOUNDS

[0050] The present invention includes defining a simple representation for compounds that captures at least some of their chemical properties, which are available from, e.g., existing sources of data. In certain embodiments, for example, a compound is denoted as x in state-space and described by a set of chemical descriptors, x_k. Thus, every compound can be placed at a point in hyperspace, which is defined by $x = (x_1, x_2, x_3, ..., x_N)$. For example, compounds can be described based on the composition of their atoms and bonds. To illustrate, the embodiment depicted in Figure 2 includes a total of 145 unique features. More specifically, based on the 145 descriptors in Figure 2, α-D-glucose (adg) is represented as the vector $x^{adg} = (0, 0, 0, 0, 0, 6, 0, 0, 0, ...)$. Similarly, pyruvate (pyr) is represented as the vector $x^{pyr} = (0, 0, 0, 0, 0, 3, 0, 0, 0, ...)$. Since the chemical descriptor space is generally large and the vector for any given compound is typically sparse, compound vectors can generally be succinctly expressed using an attribute-value notation. In this notation carbon dioxide, x^{CO2}, is described in equation 1.

$$x^{C02} = ((C\ 1)(O\ 2)(C = 0\ 2)) \qquad (1)$$

Equation 1 states that CO_2 is defined by the state vector, x^{CO2}, which contains three components: the number of carbon atoms ($x_C = 1$), the number of oxygen atoms ($x_O = 2$), and the number of C = O bonds ($x_{C=O} = 2$). The set of 145 descriptors represents most compounds uniquely. However, since chirality is not represented in this embodiment, stereoisomers map to identical points in the state-space. For example, the representation of β-D-glucose and α-D-glucose are identical. In other embodiments of the invention, descriptors representing chirality are included to account for stereoisomers.

2. TRANSFORMATIONS

[0051] To represent transformations, the complex bond changes that occur when one compound is converted to another are approximated in certain embodiments. Transformations are abstracted as transitions between compound states denoted by t. For example, the transformation of α-D-glucose (x^{adg}) into α-D-glucose-6-phosphate (x^{adg6P}) (i.e., $x^{adg} \rightarrow x^{adg6P}$) can be defined as the vector difference, which is shown in equation 2.

$$\begin{aligned} t^{adg,adg6P} &= x^{adg6P} - x^{adg} \\ &= ((C6)(H12)(O10)(P1)\ldots) - \\ &\quad ((C6)(H12)(O6)(P0)\ldots) \\ &= ((P1)(O4)(P - O\ 3)) \qquad (2) \end{aligned}$$

In equation 2, the term $t^{adg,adg6P}$ describes the state-transition as the addition of ((P1)(O4)(P - O3)). This set of descriptors corresponds to a known chemical moiety, which is the phosphate functional group (PO_4^{3-}).

[0052] Though it may not be possible to do so in all cases, the interpretation of state-transitions is often chemically intuitive. Each compound can be chemically transformed into a number of other "successor" compounds. For example, some of the chemical successors of α -D-glucose are shown in Figure 3. As described further below, the set of known chemical successors of x can be denoted as T.

[0053] Each state-transition can typically be related to a known biocatalyst. In some embodiments, for example, biochemical transformations are considered as state-transitions, t, that approximate the complex bond changes in, e.g., enzyme-catalyzed reactions. Optionally, additional biochemical attributes of reactions, like structural and energetic changes, are also included in these representations.

B. NETWORK OR PATHWAY PREDICTION AS SEARCH

[0054] After defining compounds as states and transformations as state-transitions, network or pathway inference or prediction typically becomes a state-space search problem. State-space searches have been referred to in, e.g., Artificial Intelligence (AI) research (*see, e.g.,* Pearl, Heuristics: Intelligent Search Strategies for Computer Problem Solving AddisonWesley (1984), which is incorporated by reference). In certain aspects, the present invention addresses the problem of predicting a biochemical network or pathway as searching a route from an initial or input compound to a destination or output compound through a series of state-transitions. In particular, the initial or input compound is denoted as x^0, the destination or output compound is denoted as x^L, and the network or pathway between these two is denoted as $P^{0,L}$. This is further shown in equation 3.

$$P^{0,L} = x^0 \rightarrow x^1 \rightarrow x^2 \rightarrow ...x^m... \rightarrow x^L \qquad (3)$$

[0055] The simplest approach for network or pathway inference is an uninformed search, which includes depth-first search and breadth-first search methods. In uninformed searches, successive states from x^0 are explored blindly until the goal, x^L, is reached. In real-world problems, like biochemical network or pathway searches, blind searches can lead to a combinatorially large number of possible solutions. For practical purposes it is typically desirable to reduce the set of solutions to a smaller subset. To address this issue, informed search techniques can reason over the state-space to infer network or pathways that satisfy some optimality condition. For example, a heuristic search is an informed search technique that can systematically explore a state-space by measuring the cost associated with any state-transition (Pearl (1984), *supra*). In some embodiments, informed searches take the form of best-first searches that use a heuristic evaluation function, called *F*, to reduce the combinatorially large number of possibilities faced by other methods. An exemplary simplified version of a best-first search is given in the algorithm provided in Figure 4.

[0056] More specifically, Figure 4 schematically shows a best-first search algorithm to find network or pathway, $P^{0,L}$, from an input compound, x^0, to output compound x^L, using a heuristic evaluation function, *F*. As shown, in each iteration the successors, T, of the best state in the list *X* are explored as follows. If the goal is reached then the search terminates with a path, $P^{0,L}$. Otherwise, there are two options. First, if the state x^m is not in *X*, then it is added to it using push(x^m,*X*), and a

pointer from the state to its parent is created with point(x^m,x). Second, if the state is in *X* then its heuristic score is updated to the lower out of the current and old values. Each state in *X* points to its predecessors and the path from x^0 can be traced using path(x^m). The search terminates when there are no more states to explore.

[0057] The heuristic evaluation function, *F*, can be calculated using different methods. For example, a greedy search minimizes the cost of reaching the goal state from the current state (called *H*). Conversely, a uniform cost search minimizes the cost of reaching the current state from the initial state (called *G*). Certain embodiments of the invention utilize A* (A-star) searches, which use an evaluation function that is the sum of the estimated cost thus far (*G*) and the estimated cost to the goal (*H*). In effect, this minimizes the overall path cost ($F = H + G$). To infer biochemical networks or pathways by heuristic search, certain embodiments of the present invention include a strategy for calculating the cost of a network or pathway, as described below.

1. HEUREKA

[0058] This section relates to defining the cost of state changes in a state-space representation as described herein. The biological factors that determine the cost of a network or pathway in a living system are not always known. Evolution, environment, bioenergetics, kinetics, growth, or a broader biochemical context, all may contribute to the existence of a biochemical network or pathway in an organism. The problem is that it can be difficult to calculate the contribution of most of these factors due to the scarcity of data, or the limitations of current knowledge. In certain embodiments of the invention, state-space is used to define the cost based on the chemical efficiency of a network or pathway. While this may not always be biologically correct, it is congruent with the notion that living systems tend to optimize their growth. Furthermore, it is a useful heuristic for finding synthetic networks or pathways.

[0059] To formalize the notion of cost in state-space, the difference between any two compounds is defined as Δx and the corresponding distance as $|\Delta x|$. For a state-transition this is simply $t = \Delta x$. The distance, $|\Delta x|$, can be calculated using, e.g., the Manhattan metric or the Euclidean metric, which are given in equations 5 and 4, respectively. Either the Manhattan distance or the Euclidean distance is admissible

as a heuristic, because it represents the shortest distance between any two compounds. The Manhattan distance (equation 5) is often used, because the discrete chemical changes are typically more intuitive and the computation is typically more efficient than with other distances.

$$|\Delta \mathbf{x}|_E = \sqrt{\sum_{k=0}^{k=N} (\Delta x_k)^2} \quad (4)$$

$$|\Delta \mathbf{x}|_M = \sum_{k=0}^{k=N} \Delta x_k \quad (5)$$

[0060] Using the notion of distance between states, the functions *F*, *G* and *H*, which are utilized for heuristic searches, can be evaluated. To illustrate with the hypothetical pathway shown in equation 3 (described above), which begins with the initial state, $\mathbf{x}^0$, ends with the final state, $\mathbf{x}^L$, and has any intermediate state, $\mathbf{x}^m$. The calculation of the cost functions *G* and *H* at the intermediate state, $\mathbf{x}^m$, is given in equations 6 and 7, respectively.

[0061] For an A* search the state selected for further exploration minimizes the total cost, $F = G + H$, which is shown in equation 8. Intuitively, $G(0,m)$

$$G(0,m) = \sum_{i=1}^{i=m} |\mathbf{x}^i - \mathbf{x}^{i-1}| \quad (6)$$

$$H(m,L) = |\mathbf{x}^m - \mathbf{x}^L| \quad (7)$$

$$F(0,m,L) = G(0,m) + H(m,L) \quad (8)$$
$$= \sum_{i=1}^{i=m} (|\mathbf{x}^i - \mathbf{x}^{i-1}|) + |\mathbf{x}^m - \mathbf{x}^L|$$

is the actual distance due to chemical transitions from $\mathbf{x}^0$ to $\mathbf{x}^m$, whereas $H(m, L)$ *is* a "guess" for the shortest possible distance to the goal state $\mathbf{x}^L$.

[0062] The intuition behind a heuristic search for a network or pathway includes, for example, as follows. One wants to find the series of efficient biochemical transformations that convert one compound into another. In state-space, the heuristic (*H*) is a guide for the chemical proximity of any intermediate state to the goal. By using the evaluation function in equation 8 in the algorithm schematically shown in Figure 4, one can select the network or pathway that efficiently converts the input to the output. The efficiency of this conversion is typically not determined by the length of the network or pathway. Rather, it is generally defined by an optimal value for the heuristic evaluation function, *F*. Although in certain embodiments of the invention this

function is calculated in terms of chemical distance, any biochemical property that can be calculated from available biochemical information is also optionally utilized.

[0063] The search can also be guided by, e.g., using biological information. To illustrate, networks or pathways can be searched for in an organism using a list of enzymes annotated in the genomic sequence. This can be accomplished, e.g., by modifying the algorithm schematically shown in Figure 4 to alter the successors for each state (at statement, T ← successors(x)). Since each state-transition is associated with an enzyme, the available state-transitions and allowed successors for each state are constrained by the available enzymes.

2. EVALUATING EFFICIENCY

[0064] In order to evaluate the efficiency of different search methods for computing each network or pathway one can calculate the effective branching factor, called b^*. The branching factor, called b, is the number of successors for a given state. The effective branching factor for a given computed network or pathway of length L with M nodes expanded is defined as the branching factor that a uniform tree of depth d would possess in order to contain M states. The relationship between M,d,b^* can be expressed by the polynomial given in equation 9, which can be solved numerically to estimate b^*.

$$M = \sum_{j=0}^{j=L} (b*)^j \qquad (9)$$

3. DATA

[0065] The methods, systems, and software described herein can use compound, transformation, and enzyme information from essentially any source. To illustrate, the methods, systems, and software described herein can utilize essentially any source of chemical reactions, biochemical reactions, enzyme-catalyzed reactions, metabolites/compounds, manually crafted or automatically identified biochemical transformation rules, etc. In one embodiment of the invention, for example, the Kyoto Encyclopedia of Genes and Genomes (KEGG) is used at least in part due to its accessibility and breadth of biochemical data. Parsers have been developed to import KEGG data into Lisp (the programming language in which certain embodiments of the invention are implemented). To populate biochemical state-space compound data is optionally extracted from KEGG. In certain embodiments, state-transitions are further

refined to use only, e.g., the main substrates in transformations (or main transformations). To illustrate with the following reaction:

$$Ethanol + NAD^{+} \leftrightharpoons Acetaldehyde + NADH + H^{+}$$

For example, one can map this reaction to the state-transition $Ethanol \leftrightarrow Acetaldehyde$, but not to $Ethanol \leftrightarrow H^{+}$. In some embodiments, algorithmic approaches to decomposing reactions into substrate and product relations can be utilized. In others embodiments, KEGG pathway maps or the like which already contain "main" reactions are optionally utilized. For example, this data is available from the KEGG pathway map files, which contain unordered lists of the main transformations. In addition, the KEGG genomic annotations are optionally utilized to extract the Enzyme Commission (EC) numbers for the putative enzymes in each organism. MetaCyc and other sources of functional annotation can also optionally be utilized. To further illustrate, one embodiment of the invention has data on 3,890 compounds, 2,917 transformations, and 100 organisms with annotations of putative gene functions.

C. IMPLEMENTATIONS

[0066] One embodiment of the invention has a modular and distributed architecture. There are two modes for interacting with this system: from the graphical user interface (GUI) through a web-browser, and from an interactive Common Lisp shell. The GUI is implemented as a Java client application that communicates with a Common Lisp server through TCP/IP using a custom Lisp protocol.

[0067] The server can be implemented in Allegro Common Lisp and contains modules for data management, network or pathway inference by heuristic search, visualization and distributed computing. The purpose of the data management and network or pathway inference modules are described further above. The visualization module is responsible for producing a representation of the network or pathway suitable for rendering on the client. The distributed computing module is responsible for handling all client and server interaction, and for distributing the client requests across a Lisp Parallel Virtual Machine (McShan & Shah (2002) "Lisp-PVM: Parallel Virtual Machine in Lisp for Bioinformatics. Intelligent Systems for Molecular Biology" (Poster)).

1. WEB SITE EMBODIMENT

[0068] The methods of this invention can be implemented in a localized or distributed computing environment. For example, in one embodiment featuring a localized computing environment, a system of the invention comprises a computational device equipped with user input and output features. In a distributed environment, the methods can be implemented on a single computer, a computer with multiple processes or, alternatively, on multiple computers. The computers can be linked, e.g., through a shared bus, but more commonly, the computer(s) are nodes on a network. The network can be generalized or dedicated, at a local level or distributed over a wide geographic area. In certain embodiments, the computers are components of an intranet or an internet.

[0069] In such use, typically, a client (e.g., a scientist, practitioner, provider, or the like) executes a Web browser and is linked to a server computer executing a Web server. The Web browser is, for example, a program such as IBM's Web Explorer, Internet explorer, or the like. The Web server is typically, but not necessarily, a program such as IBM's HTTP Daemon or other WWW daemon (e.g., LINUX-based forms of the program). The client computer is bi-directionally coupled with the server computer over a line or via a wireless system. In turn, the server computer is bi-directionally coupled with a website (server hosting the website) providing access to software implementing the methods of this invention. A user of a client connected to the Intranet or Internet may cause the client to request resources that are part of the web site(s) hosting the application(s) providing an implementation of the methods of this invention. Server program(s) then process the request to return the specified resources (assuming they are currently available). A standard naming convention has been adopted, known as a Uniform Resource Locator ("URL"). This convention encompasses several types of location names, presently including subclasses such as Hypertext Transport Protocol ("http"), File Transport Protocol ("ftp"), gopher, and Wide Area Information Service ("WAIS"). When a resource is downloaded, it may include the URLs of additional resources. Thus, the user of the client can easily learn of the existence of new resources that he or she had not specifically requested. To further illustrate, aspects of the invention are optionally provided as a Simple Object Access Protocol (SOAP), a Common Object Request Broker Architecture (CORBA), etc. web-service.

[0070] Methods of implementing Internet and/or Intranet embodiments of computational and/or data access processes are well known to those of skill in the art and are documented, e.g., in ACM Press, pp. 383-392; ISO-ANSI, Working Draft, "Information Technology-Database Language SQL", Jim Melton, Editor, International Organization for Standardization and American National Standards Institute, Jul. 1992; ISO Working Draft, "Database Language SQL-Part 2:Foundation (SQL/Foundation)", CD9075-2:199.chi.SQL, Sep. 11, 1997; and Cluer et al. (1992) A General Framework for the Optimization of Object-Oriented Queries, Proc SIGMOD International Conference on Management of Data, San Diego, California, Jun. 2-5, 1992, SIGMOD Record, vol. 21, Issue 2, Jun., 1992; Stonebraker, M., Editor. Other resources are available, e.g., from Microsoft, IBM, Sun and other software development companies.

Example Web Interface for Accessing Data Over a Network

[0071] Figure 5 A and B illustrate example interfaces for predicting biochemical networks or pathways using a computer interface, possibly over a web page, according to specific embodiments of the present invention. Figure 5A illustrates the display of a Web page or other computer interface for requesting a biochemical network or pathway prediction. According to specific implementations and/or embodiments of the present invention, this example interface is sent from a server system to a client system when a user accessed the server system. This example Web page contains an input selection 501, allowing a user to specify input data. As will be understood in the art, each selection button can activate a set of cascading interface screens that allows a user to select from other available options or to browse for an input file. According to specific embodiments of the present invention, option selection 502 can also be provided, allowing a user to modify the user settable options discussed herein. A licensing information section 503 and user identification section 504 can also be included. One skilled in the art would appreciate that these various sections can be omitted or rearranged or adapted in various ways. The 504 section provides a conventional capability to enter account information or payment information or login information. (One skilled in the art would appreciate that a single Web page on the server system may contain all these sections but that various sections can be selectively included or excluded before sending the Web page to the client system.)

[0072] Figure 5B illustrates the display of an interface confirming a request. The confirming Web page can contain various information pertaining to the

order and can optionally include a confirmation indication allowing a user to make a final confirmation to proceed with the order. For particular systems or analyses, this page may also include warnings regarding use of proprietary data or methods and can include additional license terms, such as any rights retained by the owner of the server system in either the data.

2. EMBODIMENT IN A PROGRAMMED INFORMATION APPLIANCE

[0073] Figure 6 is a block diagram showing a representative example logic device in which various aspects of the present invention may be embodied. As will be understood to practitioners in the art from the teachings provided herein, the invention can be implemented in hardware and/or software. In some embodiments of the invention, different aspects of the invention can be implemented in either client-side logic or server-side logic. As will be understood in the art, the invention or components thereof may be embodied in a fixed media program component containing logic instructions and/or data that when loaded into an appropriately configured computing device cause that device to perform according to the invention. As will be understood in the art, a fixed media containing logic instructions may be delivered to a viewer on a fixed media for physically loading into a viewer's computer or a fixed media containing logic instructions may reside on a remote server that a viewer accesses through a communication medium in order to download a program component.

[0074] Figure 6 shows an information appliance (or digital device) 600 that may be understood as a logical apparatus that can read instructions from media 617 and/or network port 619, which can optionally be connected to server 620 having fixed media 622. Apparatus 600 can thereafter use those instructions to direct server or client logic, as understood in the art, to embody aspects of the invention. One type of logical apparatus that may embody the invention is a computer system as illustrated in 600, containing CPU 607, optional input devices 609 and 611, disk drives 615 and optional monitor 605. Fixed media 617, or fixed media 622 over port 619, may be used to program such a system and may represent a disk-type optical or magnetic media, magnetic tape, solid state dynamic or static memory, etc. In specific embodiments, the invention may be embodied in whole or in part as software recorded on this fixed media. Communication port 619 may also be used to initially receive instructions that

are used to program such a system and may represent any type of communication connection.

[0075] The invention also may be embodied in whole or in part within the circuitry of an application specific integrated circuit (ASIC) or a programmable logic device (PLD). In such a case, the invention may be embodied in a computer understandable descriptor language, which may be used to create an ASIC, or PLD that operates as herein described.

3. INTEGRATED SYSTEMS

[0076] Integrated systems, e.g., for the methods described herein, as well as for the compilation, storage and access of databases, typically include a digital computer with software including an instruction set as described herein, and, optionally, one or more of control software, analysis software, other data interpretation software, an input device (e.g., a computer keyboard) to enter data to the digital computer, to control analysis operations, etc.

[0077] Readily available computational hardware resources using standard operating systems can be employed and modified according to the teachings provided herein, e.g., a PC (Intel x86 or Pentium chip- compatible DOS™, OS2™, WINDOWS™, WINDOWS NT™, WINDOWS 95™, WINDOWS 98™, WINDOWS 2000™, WINDOWS XP™, LINUX, or even Macintosh, Sun or PCs will suffice) for use in the integrated systems of the invention. Current art in software technology is adequate to allow implementation of the methods taught herein on a computer system. Thus, in specific embodiments, the present invention can comprise a set of logic instructions (either software, or hardware encoded instructions) for performing one or more of the methods as taught herein. For example, software for providing the biochemical network or pathway predictions can be constructed by one of skill using a standard programming language such as Common Lisp, Visual Basic, Fortran, Basic, Java, or the like. Such software can also be constructed utilizing a variety of statistical programming languages, toolkits, or libraries.

[0078] Various programming methods and algorithms, including genetic algorithms and neural networks, can be used to perform aspects of the data collection, correlation, and storage functions, as well as other desirable functions, as described herein. In addition, digital or analog systems such as digital or analog computer systems can control a variety of other functions such as the display and/or control of

input and output files. Software for performing the methods of the invention, such as programmed embodiments of the methods described above, are also included in the computer systems of the invention. Alternatively, programming elements for performing such methods as principle component analysis (PCA) or least squares analysis can also be included in the digital system to identify relationships between data. Exemplary software for such methods is provided by Partek, Inc. (St. Peter, MO); on the world wide web at partek.com.

Example System Embodiment

[0079] Figure 7 is a block diagram illustrating components that can be included in an integrated system according to specific embodiments of the present invention. This particular example embodiment optionally supports providing biochemical network or pathway predictions over a network. The server system 710 includes a server engine 711, various interface pages 713, data storage 714 for storing instructions, data storage 715 for storing, e.g., state data, state-transition data, etc., and data storage 716 for storing data generated by the computer system 710. According to specific embodiments of the invention, the server system further includes or is in communication with a processor 740 that further comprises one or more logic modules for performing one or more methods as described herein.

[0080] Optionally, one or more client systems may also comprise any combination of hardware and/or software that can interact with the server system. These systems may include digital workstation or computer systems (an example of which is shown as 720a) including a logic interface module (such as 721a) and/or various other systems or products through which data and requests can be communicated to a server system. These systems may also include laboratory-workstation-based systems (an example of which is shown as 720b) including a logic interface module (such as 721b) or various other systems or products through which data and requests can be communicated to a server system.

4. OTHER EMBODIMENTS

[0081] The invention has now been described with reference to specific embodiments. Other embodiments will be apparent to those of skill in the art. In particular, a viewer digital information appliance has generally been illustrated as a personal computer. However, the digital computing device is meant to be any

information appliance for interacting with a remote data application, and could include such devices as a digitally enabled television, cell phone, personal digital assistant, etc.

[0082] Although the present invention has been described in terms of various specific embodiments, it is not intended that the invention be limited to these embodiments. Modification within the spirit of the invention will be apparent to those skilled in the art. In addition, various different actions can be used to effect the methods described herein. For example, a voice command may be spoken by the purchaser, a key may be depressed by the purchaser, a button on a client-side scientific device may be depressed by the user, or selection using any pointing device may be effected by the user.

[0083] It is understood that the examples and embodiments described herein are for illustrative purposes and that various modifications or changes in light thereof will be suggested by the teachings herein to persons skilled in the art and are to be included within the spirit and purview of this application and scope of the claims.

III. EXAMPLES

EXAMPLE I: COMPARISONS OF SEARCH ALGORITHM EFFICIENCY

[0084] This example refers to a biochemical state-space that was built using data from known enzyme-catalyzed transformations in Ligand (Goto et al. (2002) "LIGAND: database of chemical compounds and reactions in biological pathways," Nucleic Acids Res 30(1):402-404, which is incorporated by reference), including, 2,917 unique transformations between 3,890 different compounds. To predict biochemical networks or pathways this state-space was explored using an informed search algorithm that implements a chemically motivated heuristic to guide the search. Since the algorithm does not depend on predefined networks or pathways, it can efficiently identify plausible routes using known biochemical transformations.

[0085] More specifically, this example provides the results of searching sample biochemical networks or pathways using a computer implemented embodiment of a method according to the present invention. First, the efficiencies of uninformed and heuristic searches in biochemical state-space were compared. Then, a brief overview of using a web-based system and a description of the network or pathway visualization are provided.

[0086] The efficiency of three different search algorithms in the state-space was compared including, breadth first search, depth first search and heuristic search, which are described further above. The results for four sample searches are summarized in Figure 8. The predicted networks or pathways include examples of biodegradation, biosynthesis, and biochemical engineering. For each of the network or pathway searches, the number of states explored (M), the length of the network or pathway (L), the cost (F), the effective branching factor (b*), and the computation time are provided. The computation time was measured by the Common Lisp function, called "time", which reports on the CPU usage of any computation. The timing was carried out by conducting the searches in the interactive Common Lisp shell. The timing reported in Figure 8 was measured using Allegro Common Lisp running in the RedHat Linux 7.3 operating system on a Sony PCG-CIMW laptop, containing a Transmeta Crusoe TM5800 Processor and 384 MB of main memory.

[0087] The network or pathway queries in Figure 8 cover three kinds of biochemical themes including, biodegradation (example (a)), biosynthesis (examples (b) and (d)), and engineering (example (c)). The efficiency of a web-browser, and from an interactive Common Lisp the heuristic search algorithm is described in example (a). The exploration of the network or pathway from α-D-glucose to pyruvate yielded a large number of possible solutions using blind search strategies. Figure 8 illustrates the efficiency of heuristic search in state-space over blind search strategies. Although breadth-first search finds the shortest path, it is the least efficient because it explores the largest number of states. Depth-first search is more efficient since lesser nodes are explored, but the drawback is that it produces a much longer path. The last column on the right shows the path found by heuristic search, which is the most efficient in the *F*-cost but not always the shortest. The quantitative performance measures for each of the search methods are also summarized in Figure 8. Other analyses of other network or pathway searches using these three methods have also been performed (data not shown) and the A* was the most efficient in *F* cost and in the number of states explored. Though breadth-first gives the shortest path length, L, A* is the most efficient in exploring the state-space and in optimizing the *F* cost. The effective branching factor *b** is another useful metric for comparing the searches. The breadth-first search had the highest branching factor (2.27) because it explored all immediate successors first; the depth-first search had the lowest *b** (1.28) closely followed by A* (1.38). Though the

branching factor of depth-first search is the lowest, it produces very long network or pathways, which do not seem to be very plausible. The time required for each search is roughly proportional to M.

[0088] To further illustrate, Figure 9 schematically shows the visualization of a linear network or pathway. In particular, from top to bottom the figure shows: the profile of the *F*-cost along the steps in the network or pathway; the successors of each state are shown as points; the chemical structure of the main compounds at each step; helpful statistics about each step in the network or pathway; and the EC number of the enzymes involved in catalyzing the transformations. The lower part of the visualization shows the compound names, their state descriptors, and the name of the enzyme.

EXAMPLE II: *DE NOVO* SYNTHESIS OF VANILLIN

[0089] This example illustrates aspects of the invention in the analysis of network or pathways for the *de novo* synthesis of vanillin and a transgenic strategy to implement these in a number of hosts.

A. BACKGROUND

[0090] Computational approaches to metabolic engineering generally involve quantitative calculations of kinetics, fluxes, and control coefficients to optimize a specific network or pathway. Metabolic engineering emphasizes metabolic pathway integration (Stephanopoulos (1999) "Metabolic fluxes and metabolic engineering" Metab Eng. 1(1):1–11, which is incorporated by reference). In particular, designing a novel metabolic pathway using heterologous enzymes raises many questions about the choice of enzymes and host organisms. The state of the art in gene addition and knockout is such that it is now possible to engineer almost any pathway in almost any host. Indeed, identifying the metabolic engineering strategy is usually far more complex than implementing it. Selecting a strategy typically requires not only expertise in chemistry, biocatalysis, and molecular biology, but also knowledge of what genes and enzymes are available. With the increasing amounts of genomic data, this is a challenging proposition without the use of intelligent computational aids.

[0091] An objective of this analysis was to produce a metabolic engineering strategy using only the desired chemical product, the desired carbon source and some qualitative constraints on the choice of the host organism. While the growing

databases of metabolic annotations provide a catalog of genes and enzymes, piecing them together into desired pathways is a complex task. The computational approaches described herein, which are based on heuristic search, can sift through the vast amount of information on enzymes and biotransformations to generate combinations of the necessary genes that produce biocatalytic pathways from an input to a desired product.

[0092] In this example, a pathway for *de novo* vanillin synthesis is illustrated. Vanillin, or 4-hydroxy-3-methoxybenzaldehyde, is a very important flavor and aroma molecule. More than 12,000 tons of vanillin are produced each year, but less than 1% of this is natural vanillin from the beans of the *Vanilla* orchid (Walton et al. (2003) "Vanillin" Phytochemistry 63(5):505–515, which is incorporated by reference). The natural production of vanillin is complicated by the fact that *Vanilla planifolia* flowers asynchronously, requiring hand pollination of the flowers. Additionally, the vanillin in the green vanilla beans is present exclusively in conjugated form as the β-D-glucoside, which exhibits no vanilla flavor, typically requiring extensive curing. Indeed, the worldwide increasing demand for natural vanilla flavor far outweighs the available supply with vanilla pods alone (Priefert et al. (2001) "Biotechnological production of vanillin," Appl Microbiol Biotechnol 56(3-4):296–314, which is incorporated by reference). As a result, the vast majority of vanillin is synthesized relatively cheaply via chemical processes. Thus, it is important to study alternative approaches for the production of vanillin.

[0093] Early work in *Vanilla planifolia* by Zenk (Zenk (1965) "Biosynthese von vanillin in vanilla planifola" Z.Pflanzenphysiol, which is incorporated by reference) suggested that ferulic acid might be beta-oxidized to form vanilloyl-coa and then vanillin. Other analyses indicate that the natural biosynthesis of vanillin appears to be a product of phenylalanine via the phenylpropanoid pathway, then to caffeic acid (3,4-dihydroxy-4-methoxycinnamic acid), *iso*ferulic acid (3-hydroxy-4-methoxycinnamic acid), dimethoxycinnamic acid, dimethoxybenzoic acid, and finally to vanillic acid where it is glucosylated, then reduced to vanillin (Funk et al. (1994) "Phenylpropanoid metabolism in suspension cultures of vanilla planifola" Plant Physiol., which is incorporated by reference). Vanilla biosynthesis and degradation have also been observed in several microorganisms (Achterholt et al. (2000) "Identification of Amycolatopsis sp. strain HR167 genes, involved in the bioconversion of ferulic acid to vanillin" Appl Microbiol Biotechnol 54(6):799–807, Gasson (1998)

Metabolism of ferulic acid to vanillin. A bacterial gene of the enoyl-SCoA hydratase/isomerase superfamily encodes an enzyme for the hydration and cleavage of a hydroxycinnamic acid SCoA thioester" J Biol Chem 273(7):4163–70, Mayer et al. (2001) "Rerouting the plant phenylpropanoid pathway by expression of a novel bacterial enoyl-CoA hydratase/lyase enzyme function" Plant Cell 13(7):1669–82, and Narbad et al. (1998) "Metabolism of ferulic acid via vanillin using a novel CoA-dependent pathway in a newly-isolated strain of Pseudomonas fluorescens" Microbiology 144 (Pt 5):1397–405, which are each incorporated by reference), and the relevant enzymes have been isolated from them. Since the genome for *Vanilla planifolia* has, to date, not been sequenced, these microbial enzymes can be useful for analyzing potential *de novo* vanillin biosynthetic pathways.

[0094] The biotechnological production of vanillin has focused on the biotransformation of complex feedstocks like phenolic stilbenes (Dawidar et al. (2000) "New stilbene carboxylic acid from Convolvulus hystrix" Pharmazie 55(11):848–849, Gill et al. (1987) "3,3',5'-Tri-O-methylpiceatannol and 4,3',5'-tri-O-methylpiceatannol: improvements over piceatannol in bioactivity" J Nat Prod 50(1):36–40, and Murcia et al. (2001) "Antioxidant activity of resveratrol compared with common food additives" J Food Prot 64(3):379–84, which are incorporated by reference) and eugenol (Brandt et al. (2001) "Characterization of the eugenol hydroxylase genes (ehyA/ehyB) from the new eugenol-degrading Pseudomonas sp. strain OPS1" Appl Microbiol Biotechnol 56(5-6):724–30 and Chen et al. (1988) "Isolation and characterization of an anaerobic dehydrodivanillin-degrading bacterium" Appl Environ Microbiol 54(5):1254–7, which are each incorporated by reference). See also, Walton et al. (2000) "Novel approaches to the biosynthesis of vanillin" Curr Opin Biotechnol 11(5):490–6 and Priefert et al. (2001) "Biotechnological production of vanillin" Appl Microbiol Biotechnol 56(3-4):296–314, which are both incorporated by reference.

[0095] An exemplary vanillin synthetic pathway is schematically illustrated in Figure 10 (Berry (1996) "Improving production of aromatic compounds in Escherichia coli by metabolic engineering" Trends Biotechnol 14(7):250–256 and Priefert et al. (2001) "Biotechnological production of vanillin" Appl Microbiol Biotechnol 56(3-4):296–314, which are both incorporated by reference). As shown, E4P is produced as a product of a pentose phosphate pathway, and is transformed via

the shikimic acid pathway into 3-dehydroshikimic acid which is catalyzed by the added 3-dehydroshikimate dehydratase into protocatechuic acid. Protocatechuic acid is then transformed into vanillic acid by the transgenic addition of catechol O-methyltransferase. To further illustrate, this vanillin synthetic pathway is also shown in Table I.

TABLE I

Organism	Genes	Enzyme
E.coli KL7		
	$+aroE$	shikimate dehydrogenase
	$+aroZ$	3-dehydroshikimate dehydratase
	$+P_{tac}COMT$	catechol-O-methyltransferase
	$+aroF^{FBR}$	DAHP synthetase
	$+serA$	plasmid stablizer
	$+aroB$	3-dehydroquinate synthase

B. METHODS

[0096] The *in silico* metabolic engineering approach illustrated in this example used a state-space search algorithm as described herein. More specifically, given an input metabolite, this algorithm finds an optimal biochemical pathway to a target compound.

1. THE SEARCH ALGORITHM

[0097] As referred to above, A* ("A-star") search is a well known algorithm for finding an optimal path in a graph. It works by exploring states (or nodes in a graph) in a breadth first manner according to a cost function: $f(n) = g(n) + h(n)$. In the cost function, n is a state (also called a node), $f(n)$ is the estimated cost of the cheapest solution through n, $g(n)$ is the path cost from the start state to state n. The heuristic, $h(n)$, is the estimated cost of the cheapest path from n to the goal. In addition to being both optimal and complete, A* is also optimally efficient. This means that no other optimal algorithm is guaranteed to expand fewer nodes than A* (Dechter et al. (1985) "Generalized best-first search strategies and the optimality of a*" JACM, which is incorporated by reference). The implementation of A* described in this example is similar to the one given by Russell and Norvig (Russell et al. Artificial Intelligence: A Modern Approach Prentice Hall (1995), which is incorporated by reference). The algorithm is shown in Figure 11 and is described further below. A* was specialized for

pathway searches by abstractimg metabolism as a graph in which the states are compounds and the arcs are biotransformations. Four main constructs provided for searching metabolic pathways. First, biochemical "successors" of a molecule were defined. Second, a heuristic, or an oracle, was developed for the "distance" between any molecule and the final product. Third, an evaluation function was chosen for evaluating the optimality of alternative pathways. Fourth, the condition that terminates the search on reaching the final product was defined. Each of these concepts and their implementation are described in greater detail in the following sections.

BEST-FIRST SEARCH ALGORITHM

[0098] To further illustrate, this section describes the best-first search algorithm with heuristic to find pathways from input compounds x^0 to x^L. The search optimizes f where f is the total path cost. The heuristic uses an estimate of f defined as $f = g + h$ where g is the cost of the path so far, and h is the estimate to the goal, x^L. Each transformation in the pathway has an edge cost, e. At the goal,

$$f = \sum_{i=0}^{i=L} e_i$$

This algorithm will always return the optimal cost path in terms of f if h always underestimates the actual pathcost from the current node to the goal. The algorithm begins by adding the input node, x^0 to N, the queue of nodes to be expanded. The first node on the list is popped off and stored in n. The pathway to that node is stored in $P^{0,L}$. If $P^{0,L}$ meets a selected goal test, then the pathway is returned and the algorithm is done. If not, successors are generated for the current node, returned as pairs of the reaction that performs the biotransformation and the compound to which the current node is transformed. The edge-cost, e, of the biotransformation is computed, as is the estimated distance from the successor node to the goal node, h. From these values, and the pathway cost so-far, the estimated total path length for the continuation of the current pathway, $P^{0,L}$ through the successor node is computed. A new node, n', is created for the successor, and the metrics are stored with the *reaction* and the *compound*. The successor node, n' is then pushed onto the stack of nodes to explore, N. Once all the successors are expanded, the list is resorted according to the node's estimated total cost, f. Once the list of nodes to expand is exhausted, unless a pathway is found, the algorithm ends, and returns no pathway.

2. COMPOUND SUCCESSORS: KNOWN AND NOVEL BIOTRANSFORMATIONS

[0099] The purpose of calculating successors is to generate the biocatalytically possible "next steps", or products, from any given compound. The implementation of this function takes a compound as an input and produces a list of pairs of biotransformations and output compounds. The successors of a given compound can be computed in two ways. First, by using the information about the action of enzymes on known compounds and the products of these reactions from in public databases. Second, by the prediction of novel biotransformations. Two exemplary sources of metabolic data that can be utilized are: Kyoto Encyclopedia of Genes and Genomes (KEGG) and MetaCyc. In this example the KEGG database was utilized for its breadth of organisms.

[0100] An issue in a pathway search is dealing with the combinatorial explosion due to the large number of potential successors for each compound. If the successors are restricted based on the curated information for the known biotransformations of compounds, then there are around ten successors for each compound. Since the known enzyme-catalyzed biochemistry is only a small subset of what exists in nature, there are many more biochemical successors of any given compound. To predict novel products of the action of an enzyme on a given compound, an algorithm to predict completely novel biotransformations based on a chemical graph-theoretical approach can be utilized. While this approach can identify completely novel pathways, it may be computationally intractable due to the order of magnitude increase in the number of successors of each compound. The heuristic search approaches described herein are equally powerful for harnessing the computational complexity of *de novo* pathway prediction.

[0101] Another useful supplement for curtailing the computational complexity due to the large number of successors, which may be generated either from a metabolic database or by *de novo* prediction, is a set of filters shown in Table II. Despite these filters, the combinatorial growth of the search space may still exponential. As a result, breadth first search may be computationally intractable for pathways longer than around ten steps.

TABLE II

Filter	Description
KEGG	Only generate transformation from KEGG
KEGG-directional	Only use the transformation directions annotated in KEGG
MetaCyc	Only generate transformation from MetaCyc (TBD)
avoid	Specific avoid (compounds, reactions, enzymes, genomes)
coenzyme	Ignore coenzymes
trivial	Ignore trivial molecules (H2O, CO2, NH3, etc)
inorganic	Ignore successors which do not have carbon
currency	Ignore "currency" molecules (ATP,ADP,NADPH,etc)
organism	Only use enzymes from a particular organism
annotated	Only use enzymes which are annotated with genes

3. THE HEURISTIC EVALUATION FUNCTION

[0102] In order to control the combinatorial complexity of pathway search a biochemical heuristic evaluation function, h, was utilized. It can be shown that A* is complete, optimal, and optimally efficient if the heuristic is *admissible*. An admissible heuristic is one in which $h^*(n) - h(n) \geq 0$, in which $h^*(n)$ is the actual cost from the state, n, to the goal n_{goal}. The heuristic is admissible if it never overestimates the pathway cost between two states. The condition for sub-exponential growth is $|h(n) - h^*(n)| \leq O(\log h^*(n))$. This means that the deviation of the heuristic from the actual distance must be minimized for efficient search. Figure 12 is a graph that schematically depicts combinatorial complexity.

[0103] Biochemical or chemical distance is a useful heuristic for metabolic search. This distance metric is based on a mapping of all compound states into a multidimensional feature space, C. The dimensions of C are defined by the atoms and bonds of biomolecules. The Manhattan distance between two molecules in C as the heuristic for pathway search can then be used. For any two compounds, this distance can be defined as $d(c1, c2)$. By optimizing this distance, a pathway is derived that effectively reflects the minimum number of chemical changes from the starting compound to the final compound. This is the chemically most parsimonious pathway. In general, the biochemical distance metric is a very efficient heuristic because it consistently outperforms breadth first searching by two orders of magnitude. It is also important to note that the concept of heuristic search can be used to optimize completely different cost models provided they use an admissible heuristic.

4. PREFERRING BIOTRANSFORMATION: EDGE-COST

[0104] The chemical optimality of a pathway is a useful metric for identifying a feasible metabolic route but other practical considerations can be more relevant to engineer a pathway into a host. For example, the algorithms described herein can allow the preferential selection of alternative enzymes for a transformation. To engineer a pathway in *E.coli*, for example, it may be preferable to find a pathway that requires the least heterologous enzymes. This knowledge can be provided to the algorithms described herein via the "edge-cost" function. This is called the edge-cost because it is the cost associated with an edge in the state-space search graph. For compounds *c1* and *c2*, the edge-cost is denoted by *e(c1, c2)*. To maintain the admissibility of the heuristic, the actual cost of an edge cannot be negative. That is, a pathway cost cannot be "rewarded"; but it can be "penalized." In above example, to prefer a pathway with enzymes from *E.coli* rather than other hosts, the use of a heterologous enzyme in a transformation is penalized with a positive contribution to *h*. That is, the chemical distance is used as the baseline cost, and positive cost penalties are added to it. This guarantees that the actual cost is always less than the heuristic and satisfies the requirement of admissibility. Consequently, a pathway that uses only *E.coli* enzymes will have a lower *h* than one requiring heterologous genes, rendering it more optimal.

[0105] To further illustrate, Table III shows some exemplary edge costs that have been implemented in certain embodiments of the algorithms described herein.

TABLE III

Edge Cost	Description
chemical	cost of adding/removing atoms or bonds from current node, n_{-1}, to n
step	cost per step
unannotated	cost of using an enzyme not annotated with an organism
transgenic	cost of using an enzyme organism not already in pathway
kingdom	cost of using an enzyme from a different kingdom
xenogenic	cost of using an enzyme not in Ω
elemental	cost for change in a particular element (i.e. carbon)
successors	cost for # of successors product node has in Ω (flux loss)
precusors	cost for # of precursors product node has
organic reactant	cost for other organic reactants
organic product	cost for other organic products
specific reactant	cost for using a paticular reactant (i.e. ATP)
specific product	cost for producing a particular product

[0106] One of advantages of the algorithms described herein is discovering transgenic pathways. Yet there are many practical issues in introducing and expressing multiple genes from different organism into a single host. If a specific host is desired, the algorithms described herein can associate a cost with the addition of enzymes that are not endogenous to the host. This aspect is useful, e.g., when a metabolic engineering project is targeted to a specific host. As discussed further below, if a host has not yet been selected, the algorithms described herein can make some suggestions about an appropriate choice in certain embodiments.

[0107] While the chemical or biochemical distance is a useful measure of cost, the framework described herein also allows the utilization of other cost models. In order to retain the guarantee of completeness and optimality for A*, however, one should maintain the admissibility of h. That is, h should typically underestimate the actual cost. Since it can be difficult to find heuristics for other costs, the heuristic above are commonly utilized, where $h(n) = d(n, n_{end})$, and let $e(n1, n2) = d(n1, n2) + \Sigma ei(n1, n2)$ where $ei(n1, n2)$ are the other metrics that may be desirable to optimize. This approach has been successfully implemented to minimize the use of heterologous enzymes in predicted pathways.

5. TERMINATING THE SEARCH: GOAL-TEST

[0108] The pathway search terminates when the final product has been synthesized. This is accomplished by the goal-test function, which evaluates the pathway to determine if a solution has been found. The simplest goal-test is a comparison of the last compound in the pathway with the desired product. Additional

pathway selection criteria like path length, maximum pathway cost, maximum number of heterologous enzyme, presence or absence of a specific intermediate, etc. can also be specified.

C. RESULTS

[0109] The heuristic search the algorithm described above was used to computationally derive a metabolic strategy for the *de novo* synthesis of vanillin from d-glucose in *E.coli*. The KEGG annotation of transformations (essentially searching through the pathway maps) was utilized. This produced a 19 step pathway as shown in Table IV, where f is the estimated pathway length ($f = g + h$), g is the exact path length so far, and h is the heuristic estimate of the distance to the goal. Both g and h were computed using distance in chemical space, C. The number of organisms which code for each enzyme is represented by $|\Omega|$, and shown in the last column.

TABLE IV

EC	Enzyme	Compound	f	g	h	$\|\Omega\|$
		alpha-d-glucose			22	
5.3.1.5	xylose isomerase	levulose	22	0	22	26
2.7.1.1	hexokinase	β-D-fructose 6-phosphate	42	10	32	69
2.2.1.1	transketolase	D-erythrose 4-phosphate	52	29	23	111
2.5.1.54	3-deoxy-7-phosphoheptulonate synthase	3-deoxy-arabino-heptulonate 7-phosphate	80	49	31	98
4.2.3.4	3-dehydroquinate synthase	5-dehydroquinate	84	66	18	97
4.2.1.11	phosphopyruvate hydratase	3-dehydroshikimate	86	74	12	222
4.2.1.10	3-dehydroquinate dehydratase	protocatechuate	94	84	10	104
4.1.1.63	protocatechuate decarboxylase	catechol	104	92	12	0
1.3.1.55	cis-1,2-dihydrocyclohexa-3,5-diene 1-COOH dehydrogenase	cis-1,2-dihydroxycyclo-3,5-diene-1-COOH	112	104	8	0
1.14.12.10	benzoate 1,3 dioxygenase	benzoate	122	114	8	1
1.14.13.12	benzoate 4 monooxygenase	4-hydroxybenzoate	126	118	8	0
4.1.1.161	4-hydroxybenzoate decarboxylase	phenol	138	126	12	1
4.1.99.2	tyrosine phenol lyase	l-tyrosine	160	148	12	3
4.3.1.5	phenylalanine ammonia lyase	4-coumarate	164	158	6	2
1.14.18.1	monophenol monooxygenase	trans-caffiate	170	162	8	5
2.1.1.68	caffeate o-methyltransferase	ferulate	180	170	10	0
6.2.1.34	trans-feruloyl-CoA synthase	feruloyl-coA	510	335	175	0
4.2.1.101	trans-feruloyl-CoA hydratase	3-OH-3,4-OH-3-methoxyphenylpropionyl-CoA	524	343	181	0
4.1.2.41	vanillin synthase	vanillin	524	524	0	0

To further illustrate, Figure 13 shows the net equation, including all side compounds, for this pathway.

[0110] Given the magnitude of the search space, automatically identifying this pathway is an important result. An alternative pathway search engine from KEGG, called PathComp (Ogata (1998) "Computation with the kegg pathway database" BioSystems, which is incorporate by reference), was unable to return a solution for this search. This was probably because it implements a breadth first search algorithm, which suffers from the combinatorial explosion mentioned above. Based on the exponential nature of the time complexity encountered in pathway search (tms = $0.31 * 4.74^{19} = 1 \times 10^{13}$ *ms* = 67.5 *yrs*), it can take up to 70 years to explore all of the pathways with a length of 19 steps using a standard breadth-first search method. The

algorithm used in this analysis took about 18 seconds to find the optimal solution. As knowledge of biotransformations grows, the time complexity of predicting biochemical routes increases exponentially making intelligent search a necessary tool for *in silico* pathway engineering.

[0111] With this 19 step predicted pathway for *de novo* vanillin synthesis, this algorithm can also aid in designing a metabolic engineering strategy through a number of tools. First, it can identify a suitable host for engineering this pathway. It accomplishes this by sorting the list of all known organisms by the number of genes they have for encoding the enzymes in the desired pathway. For vanillin synthesis this turned out to be a tie between *Brucella melitensis* and *Streptomyces coelicolor* with eight of the genes present in each organism. Though these organisms are not common hosts, they can be worth considering. *E.coli* 0157 has seven of the necessary genes, and the transgenic strategy for this pathway is shown in Table V.

TABLE V

E.coli +

Organism	Genes	Enzyme
M.loti MAFF303099 *S.meliloti 1021* *A.thaliana*	+*tyrC*	cyclyhexadienyl dehydrogenase
... *A.thaliana*	+AT2G37040	phenylalanine ammonia lyase
H.sapiens, *M.musculus* *D.melanogaster* *et.al.*	+*tyr*	monophenol monooxygenase
* soybean, spinach	?	caffeate o methyltransferase
* *P.fluorescens* AN103[15] * *S.viridiosporus*[17]	?	trans feruloyl CoA synthase
* *P.fluorescens* AN103[15] * *S.viridiosporus*[17]	?	trans feruloyl CoA hydratase
* *P.fluorescens* AN103[15] * *S.viridiosporus*[17]	?	vanillin synthase

It would take seven heterologous genes to transform L-tyrosine to vanillin via ferulate. This pathway has been discussed at great length in, e.g., Priefert et al. (2001) "Biotechnological production of vanillin" Appl Microbiol Biotechnol 56(3-4):296–314 and Walton et al. (2003) "Vanillin" Phytochemistry 63(5):505–515, which are both incorporated by reference, and *V. planifolia* uses a variant of this pathway. However, the genes encoding the enzymes for the biotransformation of ferulate to vanillin are not annotated in KEGG. This is not surprising since KEGG only contains the annotations

for gene from complete genomes. One of the advantages of KEGG and MetaCyc is that they maintain literature references indicating which the organisms that have been observed to code for these enzymes.

[0112] The optimal pathways for several other host options (*B.melitensis* (BME); *S.coelicolor* (SCO); *E.coli.* (ECO); and *A.thaliana* (ATH)) are illustrated in Figure 14 and Table VI.

TABLE VI

	Organism	1	2 BME	3 SCO	4 ECO	5 ATH
	R	19	19	20	20	21
	N	1233	1257	1216	1246	1075
	B*	1.4	1.4	1.3	1.3	1.3
	E	524	7526	6532	7526	4532
Abbr/EC	Compound/Enzyme					
ADGLU	ALPHA-D-GLUCOSE	o	o	o	o	o
5.3.1.5	XYLOSE_ISOMERASE	1	⇓	⇓		
5.1.3.3	MUTAROTASE				⇓	⇓
BDGLU	BETA-D-GLUCOSE				o	o
2.7.1.1	HEXOKINASE				1	⇓
BDG6P	BETA-D-GLUCOSE_6-PHOSPHATE				o	o
5.3.1.9	OXOISOMERASE				⇓	⇓
LEV	LEVULOSE	o	o	o		
2.7.1.1	HEXOKINASE	1	1	1		
BDF6P	BETA-D-FRUCTOSE_6-PHOSPHATE	o	o	o	o	o
2.2.1.1	TRANSKETOLASE	1	⇓	⇓	o	o
ERY4P	D-ERYTHROSE_4-PHOSPHATE	o	o	o	o	o
2.5.1.54	DS-MN	1	⇓	⇓	o	⇓
3DAH7P	3-DEOXY-ARABINO-HEPTULONATE_7-PHOSPHATE	o	o	o	o	o
4.2.3.4	CYCLIZING	1	⇓	⇓	⇓	⇓
5DHQ	5-DEHYDROQUINATE	o	o	o	o	o
4.2.1.10	DHQASE					
4.2.1.11	ENOLASE	1	⇓	⇓	⇓	⇓
3DHS	3-DEHYDROSHIKIMATE	o	o	o	o	o
4.2.1.10	DHQASE	1				
1.1.1.25	SHIKIMATE_OXIDOREDUCTASE		⇓	⇓	⇓	
1.1.99.25	NAD-P-INDEPENDENT_QUINATE_DEHYDROGENASE					1
SHI	SHIKIMATE		o	o	o	o
2.7.1.71	SHIKIMATE_KINASE		⇓	⇓	⇓	⇓
PRO	PROTOCATECHUATE	o				
4.1.1.63	PROTOCATECHUATE_DECARBOXYLASE	1				
SH5P	SHIKIMATE_5-PHOSPHATE		o	o	o	o
2.5.1.19	EPSP_SYNTHASE		⇓	⇓	⇓	⇓
CAT	CATECHOL	o				
1.3.1.55	2-HYDRO-1-2-DIHYDROXYBENZOATE_DEHYDROGENASE	1				
05I3P	05-1-CARBOXYVINYL-3-PHOSPHOSHIKIMATE		o	o	o	o
4.2.3.5	CHORISMATE_SYNTHASE		⇓	⇓	⇓	⇓
CIS12DIHC3SD1C	CIS-1-2-DIHYDROXYCYCLOHEXA-3-5-DIENE-1-CARBOXYLATE	o				
1.14.12.10	BENZOIC_HYDROXYLASE	1				
CHO	CHORISMATE		o	o	o	o
5.4.99.5	CHORISMATE_MUTASE		⇓	⇓	⇓	⇓
BEN	BENZOATE	o				
1.14.13.12	BENZOIC_4-HYDROXYLASE	1				
PRE	PREPHENATE		o	o	o	o
2.6.1.57	ARAT		1		⇓	1
4.2.1.51	PREPHENATE_DEHYDRATASE			⇓		⇓
4HBEN	4-HYDROXYBENZOATE	o				
4.1.1.61	P-HYDROXYBENZOATE_DECARBOXYLASE	1				
PHE	PHENYLPYRUVATE			o		o
2.6.1.57	ARAT			1		1
PHE	PHENOL	o				
4.1.99.2	BETA-TYROSINASE	1				
LPHE	L-PHENYLALANINE			o		o
1.14.16.1	PHENYLALANINASE			1		
4.3.1.5	PAL					⇓
PRE	PRETYROSINE		o		o	
1.3.1.43	AROGENIC_DEHYDROGENASE		⇓		1	
TRACIN	TRANS-CINNAMATE					o
1.14.13.11	C4H					⇓
LTYR	L-TYROSINE	o	o	o	o	
4.3.1.5	PAL	1	1		1	
1.14.18.1	CRESOLASE			⇓	1	
4COU	4-COUMARATE	o	o		o	o
1.14.18.1	CRESOLASE	1	1		1	1
LDOP	L-DOPA			o		
4.3.1.11	DIHYDROXYPHENYLALANINE_AMMONIA-LYASE			1		
TRACAF	TRANS-CAFFEATE	o	o	o	o	o
2.1.1.68	CAFFEATE_METHYLTRANSFERASE	1	1	1	1	
6.2.1.12	4CL					⇓
CAFCOA	CAFFEOYL-COA					o
2.1.1.104	CAFFEOYL-COA_O-METHYLTRANSFERASE					⇓
FER	FERULATE	o	o	o	o	
6.2.1.34	TRANS-FERULOYL-COA_SYNTHASE	1	1	1	1	
FERCOA	FERULOYL-COA	o	o	o	o	o
4.2.1.101	TRANS-FERULOYL-COA_HYDRATASE	1	1	1	1	1
3H34H3MCOA	3-HYDROXY-3-4-HYDROXY-3-METHOXYPHENYLPROPIONYL-COA	o	o	o	o	o
4.1.2.41	VANILLIN_SYNTHASE	1	1	1	1	1
VAN	VANILLIN	o	o	o	o	o

[0113] One of the features of certain embodiments of the algorithms described herein is the ability to interactively search for pathways. One such feature is the ability to control the directionality of biotransformations. For example, when the

constraint on the directionality of transformations from KEGG was relaxed, a biochemically optimal pathway from alpha-d-glucose to vanillin through protocatechuic acid was found. This is a much shorter route, and is a compelling result because it is the only known example of *de novo* synthesis of vanillin from glucose in *V.planifolia* (Priefert et al. (2001) "Biotechnological production of vanillin" Appl Microbiol Biotechnol 56(3-4):296–314, which is incorporated by reference). The algorithm described above suggested that *B. japonicum* has the distinction of "best host" for this pathway. This pathway is nearly identical to the *de novo* strategy discussed above and illustrated in Figure 10 and referred to in Table I.

[0114] When the KEGG annotation of transformations was used, a nineteen step pathway through ferolic acid was found. This is because reactions in KEGG do not include the transformation of protocatechuate to vanillin. The reaction is annotated as bidirectional "⇔", but only the reverse transformation from vannillate to protocatechuate is annotated in KEGG. When the directionality constraint is relaxed, it may be possible to transform protocatechuate ⇒ vanillin via vanillate demethylase. However, the *E.coli* engineering discussed in literature used catechol-methyltransferase (EC 2.1.1.6) and not the vanillate demethylase (EC 1.2.3.12) described in the pathway solution provided above.

[0115] Relaxing the directionality can be rationalized, because there are large amounts of missing annotation in the metabolic databases, and most biotransformations are reversible (though this may involve different enzymes). This turned out to be a valid assumption in this case as vanillate demethylase catalyzes demethylation or monohydroxylation with a variety of different aromatic substrates (Morawski et al (2000) "Substrate range and genetic analysis of acinetobacter vanillate demethylase" J. Bacteriology,, which is incorporated by reference). Ultimately, this is a limitation of the annotation: the algorithm described above would have used the catechol o-methyltransferase if it had been annotated. KEGG does have catechol o-demethylase genes in the human, the mouse and the rat genomes but it is not functionally characterized as an enzyme responsible for the biotransformation of protocatechuic acid to vanillic acid.

[0116] This is not limited to just a *de novo* fermentation approach. There is, for example, a reported pathway in white rot fungi for the degradation of the guaiacylglycerol-β-coniferyl ether unit of lignin to vanillin (Ishikawa et al. (1963)

"Investigations on lignins and lignification. xxviii. the enzymaticdegradation of softwood lignin by white rot fungi" Arch Biochem Biophys, which is incorporated by reference). Lignin is one of the most abundant natural sources of aromatic compounds, and it is an obvious source for vanillin biosynthesis. In fact, there is already a process for the chemical oxidation of lignin (Clark (1990) "Vanillin" Perf Flavor, which is incorporated by reference). In KEGG, lignin only occurs in the context of "Flavonoid, Stilbene and Lignin Biosynthesis" as a terminal product (i.e., all transformations lead to lignin but none use it as a precursor). As a result, the embodiment of the algorithm described above could not find a pathway from lignin to vanillin. However, when the directionality was relaxed, a quick pathway search from lignin to vanillin found the pathway from lignin through coniferol, shown in Figure 15, which is converted to coniferyl aldehyde (ferulaldehyde) that joins the above pathways at ferulate. While the white-rot fungi are not genomically annotated, a pathway can be found based on literature curation.

[0117] In summary, the above analysis shows, e.g., how the heuristic search algorithms of the invention can elucidate a transgenic strategy for metabolic engineering of vanillin from d-glucose using the KEGG database. The pathway is nineteen enzymatic steps in length, and as such it cannot be solved by other computational approaches. The chemical space heuristic with cost penalties allows one to precisely define the criteria that are important for the solution. As more and more genomes are sequenced, and novel enzymes are annotated, the algorithms described herein will become increasingly useful for metabolic engineering among many other applications.

EXAMPLE III: SYMBOLIC INFERENCE OF XENOBIOTIC METABOLISM

[0118] This example relates to *de novo* pathway inference and its practical application to a biomedical problem: xenobiotic metabolism.

A. INTRODUCTION

[0119] With the availability of the complete genomic blueprint for living systems and a large set of known biotransformations, it is becoming possible to theoretically elucidate metabolism. This includes the analysis of endogenous as well as xenobiotic pathways. Drugs, substances of abuse and environmental pollutants are examples of compounds that may not occur naturally in a living system. Since these

compounds and/or their metabolic by-products can be potentially toxic, investigating xenobiotic metabolism is important for human health and the environment. The prediction of xenobiotic metabolism is also described in, e.g., Langowski et al. (2002) "Computer systems for the prediction of xenobiotic metabolism" Adv. Drug Deliv. Rev. 54(3):407-415, which is incorporated by reference.

[0120] The metabolic potential of a living system depends on biocatalysis. However, understanding the mechanisms of enzymatic catalysis is an extremely difficult problem, and knowledge in this area is limited to a handful of well studied examples. Generally, empirical "rules" for the biotransformation of metabolites by enzymes can be abstracted. For instance, consider the broad range of substrates for *Saccharomyces cerevisae* (yeast) alcohol dehydrogenase (YADH), which reduces acetaldehyde and a variety of other aldehydes (Applications of Biochemical Systems in Organic Chemistry, Wiley (1976), which is incorporated by reference), and oxidizes ethanol, and other acyclic primary alcohols. Yet an alcohol dehydrogenase from *Thermoanaerobium brokii* (TADH) catalyzes the stereospecific reduction of ketones and the oxidation of secondary alcohols. The functions of YADH and TADH share common attributes and have some unique differences They are both alcohol dehydrogenases but their specificities for the alcohols are different. The functions of these enzymes can be expressed in terms of the functional groups modified (alcohol to aldehyde or ketone), and the backbone structure of the molecule (primary or secondary alcohol). This is essentially a symbolic description of biocatalysis and as described herein, it can be applied to complete metabolic systems.

B. METHODS

[0121] The strategy for elucidating *de novo* xenobiotic metabolism in this example consists of two main steps. First, biotransformation data was used to derive symbolic chemical substructural rules that generalized the action of enzymes on specific compounds. Second, these rules were applied iteratively to a compound to generate a plausible metabolic system. These steps and the metabolic representation are described in the following sections.

1. REPRESENTING BIOTRANSFORMATIONS AND RULES

[0122] The abstraction of metabolic concepts in this example relates to work by Karp, *supra*, in terms of the high-level concepts including pathways, enzyme-

catalyzed reactions and transformations. At the level of biotransformations, the focus is on specific chemical substructural details of metabolites that are modified through biocatalysis (Kazic, Reasoning about biochemical compounds and processes, pp. 35-49, World Scientific (1992), which is incorporated by reference). In this analysis, compounds are represented as *X*. Compounds in this abstraction have a chemical structure, which is represented as a molecular graph, Γ, in which nodes are atoms and edges are bonds. In the context of a biotransformation, the pattern of substructural changes from the input compound to the output compound is represented as a rule, *U*. A rule captures the concept of functional group changes that occur in a biotransformation. Rules are implicitly unidirectional so reversible transformations are represented as two separate rules. The two molecular graphs of a rule are indicated by the input graph, Δ^-, and the output graph, Δ^+. For example, the rule for the conversion of a primary alcohol to an aldehyde is shown in Figure 16, where Δ^- is an alcohol moiety, which is converted to Δ^+, an aldehyde moiety.

[0123] This analysis focuses on changes at the level of functional groups between pairs of compounds. The conversion of one input compound to one output compound is represented as a transformation. This simplifies the representation of reactions in terms of the main metabolites. This data was obtained from the KEGG distribution, but automated methods for identifying the main metabolites in a reaction can also be utilized.

2. EXTRACTING TRANSFORMATION RULES FROM REACTION DATA

[0124] One strategy for identifying rules is to curate them manually, however, an objective of this analysis was to use the available metabolic data to derive biotransformation rules automatically (Karp et al. (1999) "Integrated pathway/genome databases and their role in drug discovery" Trends in Biotechnology 17(7):275-281, which is incorporated by reference). This is a difficult problem in general as the information about reactive moieties is not explicitly available. In this example, a simple strategy was used for extracting rules automatically from "general" reactions. In KEGG, for instance, general reactions are defined when the input and the output compounds are both Markush structures. 741 general reactions were found in KEGG, which constitute 20% of the reactions annotated as being human-specific. For example, a gene that is extremely important in xenobiotic metabolism and encodes cytochrome

P-450 enzyme, CYP2D6, is implicated in the disposition of over thirty toxins. In KEGG, the P-450 enzyme (EC 1.14.14.1) is associated with only four reactions as shown in Figure 17. There are two specific reactions involved in endogenous functions associated with tryptophan metabolism and gamma-hexachlorocyclohexane degradation. The other two operate on general compounds denoted by their Markush structures (these are abstract structures containing a wildcard "R" group and specific functional groups). These general reactions were automatically converted to rules as described above. This was done by replacing the wildcard of the substrate with "C" and storing it as the Δ^- subgraph in the resulting rule; similarly, the "R" in the product graph is replaced and the resulting graph is stored as Δ^+.

[0125] In this example, the focus was on the rules important in xenobiotic metabolism in mammalian systems, including oxidation, reduction, hydrolysis and conjugation to mention a few. There are generally two phases in xenobiotic metabolism. In phase 1 the compounds are 'functionalized', which means that a reactive functional group is exposed. Detoxification occurs in phase 2 by further action on the functional groups, which is the form in which the compound is excreted. For instance, the first phase may activate a molecular oxygen in the input compound, and the second phase may conjugate it. Glucuronidation is the most common conjugate and can be attached to any labile oxygen. In the case of alcohol metabolism, both the alcohol and the acid can usually be conjugated.

3. BIOTRANSFORMATION RULE APPLICATION

[0126] The rule application algorithm is illustrated in Figure 19, which is also described further below. A rule is applied to a substrate X_s by searching the graph of X_s, Γ_s for the subgraph Δ^-. If the subgraph Δ^- is found, it is replaced by the Δ^+ graph to yield the product graph, Γ_p. This is summarized as follows:

$$\Gamma_s - \Delta^- + \Delta^+ \Rightarrow \Gamma_p.$$

This is also graphically illustrated in Figure 18.

[0127] The product of applying a rule to a compound can be a completely novel compound or a known compound. A subgraph isomorphism was used to search the product molecular graph against the database of known compounds. If the compound was not found, a novel compound X'_p was created and given a unique identifier (Nxxxxxx in which x was a digit from 0-9). The corpus of all rules was

designated $\bar{U}$. The top-level function is metabolize($X,\bar{U},n$) which takes a compound X and systematically applies each rule in the rule-base $\bar{U}$ through n iterations.

ALGORITHM

[0128] The metabolize($X,\bar{U},n$) algorithm creates a network of pathways length n from input compound X_s by applying rules $\bar{U}$. Initially the list of *Products* is set to null. The molecular graph, Γ_s, of the input compound was obtained from the KEGG mol file representation. For every rule in the rulebase $\bar{U}$, Δ^- and Δ^+ subgraphs are obtained. The product graph, Γ_p is obtained by performing a graphical search/replace on the input graph, Γ_s. If Γ_p is non, i.e., a match was found and applied, then the product graph Γ_p is searched against the database of known compounds and the database of novel compounds to see if an isomorphic graph exists. If the graph matches an existing compound, then X_p is returned. If there is no identified compound with the graph, then a novel compound, X_p is generated and given a unique identifier (the Nxxxxx symbols described above). In either case, the product, X_p is pushed onto the Products list for this metabolite X_s. This process can occur iteratively for every product, X in the Products list. The metabolize function is simply called again with the recursion level reduced. The results are appended to the Products list.

4. IMPLEMENTATION

[0129] The system utilized in this analysis was implemented in Allegro Common Lisp. The metabolic databases were read in and parsed into CLOS structures. For visualization, the transformations were exported to the AT&T graphviz program neato which did a simple force-based layout of the metabolic graph. This network was read back in and presented with the nodes replaced by compound structures using an internal visualization system. The novel compounds that were produced by the application of the rules were simply graphs. In order to visualize the compounds, 2D coordinates were utilized. To achieve this, the graph as a mol file was exported with the 2D coordinates as zeroes and then the mol file was laid out using the JChem molconvert package. The mol files were read back in and stored with the compounds as they are created.

C. RESULTS

[0130] The version of the KEGG database utilized had 10,635 compounds, out of which 825 were generic. Of the 5,428 reactions in the KEGG

database, 741 operated on the generic compounds. From this data, 110 biotransformation rules were inferred, and the 10 simplest ones are summarized in Table VII.

TABLE VII

U_R	Reactant	Product	E.C.	Enzyme
1	ALCOHOL ROH	β-L-ARABINOSIDE $RC_5H_9O_5$	3.2.1.88	VICIANOSIDASE
2	ALKYL SULFATE RO_4HS	ALCOHOL ROH	2.3.1.84	ALCOHOL ACETYLTRANSFERASE
3	ALCOHOL ROH	ACETYL ESTER $RC_2H_3O_2$	2.3.1.84	ALCOHOL ACETYLTRANSFERASE
4	ALCOHOL ROH	GLUCURONIDE $RC_6H_9O_7$	3.2.1.31	KETODASE
5	FATTY ACID $RCHO_2$	α-OH FATTY ACID $RC_3H_5O_3$	1.14.14.1	MICROSOMAL P-450
6	R-CN RCN	MCA AMIDE RCH_2NO	4.2.1.84	NHASE
7	1-ALCOHOL RCH_3O	ALDEHYDE $RCHO$	1.1.99.20	ALKAN-1-OL DEHYDROGENASE
8	ALDEHYDE $RCHO$	FATTY ACID $RCHO_2$	1.2.99.3	ALDEHYDE DEHYDROGENASE
9	ALDEHYDE $RCHO$	R-COOH RCO_2	1.2.3.1	ALDEHYDE OXIDASE
10	R-COOH RCO_2	ALDEHYDE $RCHO$	1.2.3.1	ALDEHYDE OXIDASE

These rules correspond to enzymes, which have flexibility in the substrates they can transform.

[0131] Using the symbolic computational approach described above the *de novo* metabolism of two compounds was elucidated. First, ethanol is one of the compounds that was considered. It is a common substance of abuse and there is some data of human metabolism. Second, the fate of furfuryl alcohol was demonstrated. Furfuryl alcohol is an industrial organic solvent used as a paint thinner and was absent in the database utilized. Experimental evidence suggests that prolonged exposure to furfuryl alcohol may have significant toxicological effects. The rules were first applied to ethanol, which was in the database. The graph is shown in Figure 20. Next, the rules were applied to the new compound, furfuryl alcohol. The result is shown in Figure 21. Some of the nodes in the ethanol metabolism graph matched to known compounds in the database. Additionally, the pathway, *alcohol* $\Rightarrow$ *aldehyde* $\Rightarrow$ *acid* $\Rightarrow$ *conjugation*, was identified. It recapitulates the standard ethanol detoxification pathway. Metabolites for a compound previously unknown to the system were also predicted. The furfuryl alcohol metabolic predictions were consistent with literature. For example, Martin et. al. (Martin et al. (2002) "General O-glycosylation of 2-furfuryl alcohol using beta-glucuronidase" Biotechnot Bioeng 80(2):222-7, which is incorporated by reference), report that furfuryl alcohol can be O-glycosylated by beta-Glucuronidase as predict by this analysis (shown as compound N00482 in Figure 21). Additionally, the acid offurfurol, 2-furoate, was actually in the KEGG database and was identified as such by the algorithm. Nomeir et. al., for example, report that the

initial step in furfuryl alcohol metabolism in rat is the oxidation to furoic acid, which is excreted unchanged and decarboxylated, or conjugated with glycine or condensed with acetic acid (Nomeir et al. (1992) "Comparative metabolism and disposition of furfural and furfuryl alcohol in rats" Drug Metab Dispos 20(2):198-204, which is incorporated by reference). The limitations of the system utilized in this example to predict the condensation with acetic acid, for instance, lie in the breadth of the rules, not in the fundamental methodology. By extending the method for inferring new rules based on known biochemistry this limitation can be overcome.

[0132] Most of the complex products of furfuryl alcohol were simply consecutive glucurodinations by the rule:

$$\text{Alcohol} \Rightarrow \beta\text{-D-Glucuronide}$$

[0133] Due to the lack of specificity of this rule to primary alcohols, glucuronidation is applied to the hydoxyl groups on the β-D-Glucuronide. While this might be biologically valid, in reality, glucuronidation renders a compound water soluble after which it is eliminated by excretion. This limitation was beyond the scope of this analysis but can be addressed by considering the physical properties of compounds, like water-solubility.

[0134] That a biotransformation rule can be applied does not imply that it is biochemically valid. For instance, consider the biotransformation rules that apply to a hydroxyl functional group. Compounds containing this functional group include primary alcohols, secondary alcohols, and also carboxylic acids. Enzymes that act on alcohols may not act on carboxylic acids and vice-versa. To capture the substrate specificity of enzymes other representations of rules can improve their biological validity. Nonetheless, the predictions generated using the algorithm described in this example are useful for elucidating potential xenobiotic metabolism, which can be tested experimentally.

[0135] It is important to contrast the approach described in this example with other rule-based approaches to pathway prediction, such as the one described in Hou et al. (2003) "Microbial pathway prediction: a functional group approach" J Chem Inf Comput Sci 43(3):1051-7, which is incorporated by reference. One of the advantages of the strategy described herein is automated biotransformation rule extraction from available resources of metabolic data. As opposed to the manual curation-based efforts, the approach described in this example scales elegantly with

increasing data for at least two significant reasons. First, the algorithm described in this example for rule extraction can be extended to utilize most of the available enzyme-catalyzed reaction data beyond the generic reactions in KEGG. Second, the combinatorial explosion of plausible biotransformations can be controlled by extending the algorithm on pathway search. Another advantage is that biotransformation predictions are related to the organism-specific enzymes and genes, which is typically important for *in vivo* or *in vitro* experimental validation.

[0136] In summary, this example illustrates aspects of a symbolic inference approach of the invention and demonstrates the *de novo* elucidation of metabolism. This was accomplished by representing biocatalysis, which is the basis of metabolism, in terms of expressive symbolic biotransformation rules. These biotransformation rules generalize the biocatalytic functions of enzymes and enable the discovery of new metabolic potential in living systems. Certain embodiments of the algorithm extract these rules from known enzyme-catalyzed reactions and apply these rules to elucidate the metabolism of new compounds. This concept was successfully tested to predict the xenobiotic metabolism of ethanol and furfuryl alcohol. These results were encouraging particularly because furfuryl alcohol was absent from the database utilized and yet its products were correctly identified through O-glycosidation and oxidation to furoic acid in agreement with the literature. These results are also biologically interesting because they support the notion that xenobiotic metabolism is a manifestation of endogenous biocatalytic abilities in an organism. Further, the method is quite general and scalable for investigating the metabolic network of any living system. These symbolic approaches to discovering the biochemical capabilities of living systems can be also utilized in combination with high-throughput or traditional experimental strategies.

[0137] While the foregoing invention has been described in some detail for purposes of clarity and understanding, it will be clear to one skilled in the art from a reading of this disclosure that various changes in form and detail can be made without departing from the true scope of the invention. For example, all the techniques and apparatus described above may be used in various combinations. All publications, patents, patent applications, or other documents cited in this application are incorporated by reference in their entirety for all purposes to the same extent as if each

individual publication, patent, patent application, or other document were individually indicated to be incorporated by reference for all purposes.

APPENDIX

INTERNATIONAL PATENT APPLICATION For

METHODS, SYSTEMS, AND SOFTWARE FOR PREDICTING BIOCHEMICAL PATHWAYS

```
(eval-when (compile load eval)
  (cl-user::use :cl-user)
  (cl-user::use :comp.search.aima)
  (cl-user::use :utils.utilities)
  (cl-user::use :math.newton)
  (cl-user::use :bio.db.pathminer.pathminer))

(provide :bio.db.pathminer.pathway)

(in-package pathminer)

(use-package :comp.search.aima)

(cl-user::use :utils.defstruct-clos)

; Pathway  definition and results
(defstruct-clos pathway
  ; definition
  id
  name
  from
  to
  algorithm
  algorithm-parameters

  debug
  backwards
  compounds
  reactions
  organism
  maximize
  minimize
  add
  remove
  allow
  avoid
  prefer
  predict
  require-organisms
  require-enzymes
  organic-reactant-cost
  organic-product-cost
  carbon-cost
  successor-cost
  xenogenic-cost
  unannotated-cost
  step-cost
  distance-cost
  kingdom-cost
  in-space ; Atom,Bond,Chemical
  distance
  goal-test
  min-steps
  max-steps
  results)

 (defstruct-clos pathway-result
   pathway
   problem
   solution
```

```
  compounds
  reactions
  reactants
  products
  organism
  organisms
  N
  d
  b*
  f

  expanded-transformations ; (compound . reaction)
  transformations
  critical)

(defmethod pathway-result-reactions ((pr pathway-result))
  (mapcar #'symbol-value (slot-value pr 'reactions)))

(defmethod pathway-result-compounds ((pr pathway-result))
  (mapcar #'symbol-value (slot-value pr 'compounds)))


;;;
;;; AIMA Definitions
;;;

; uncomment for first load

;(eval-when (compile load)
;  (defvar *aima-loaded* nil)
;  (unless *aima-loaded*
;    (cl-user::use :pub.aima.aima)
;    (aima-compile 'search)
;    (aima-load 'search)
;    (setq *aima-loaded* t)))

(defstruct (pathway-finding-problem (:include problem)))

(defvar *pathway* nil)
(defvar *problem* nil)
(defvar debug nil)


(defun print-rc (rclist)
  (unless (listp rclist) (setq rclist (list rclist)))
  (dolist (rc rclist)
    (format t "~A ~A~%" (reaction-id (car rc)) (compound-id (cdr
rc)))))

;;
;; cache expansion in hash by compound id
(defmethod successors ((problem pathway-finding-problem) x)
  (let (node compound successors organism
             (goal (problem-goal problem))
             (predict (pathway-predict *pathway*)))
    (cond ((node-p x) (setq node x) (setq compound (node-state node)))
          ((compound-p x) (setq compound x)))

    (cond (use-kegg
           (cond ((pathway-backwards *pathway*)
```

```
              (setq successors (compound-lkup-transformations-
precursors compound :rc t)))
             (t
              (setq successors (compound-lkup-transformations-
successors compound :rc t))))
          )
         (t
          (setq successors (compound-rc-successors compound))
          ))


    (when predict
      (let ((predictions (funcall predict compound)))
      (format t "predictions: ~A~%" predictions)))

    (dolist (successor successors)
      (unless (AND (reaction-p (car successor))
                   (compound-p (cdr successor)))
      (break)))

    (when debug
      (format t "~A initial successors:~% " (length successors))
      (dolist (successor successors)
      (format t "~A " (compound-id (cdr successor))))
      (format t "~%"))

    (setq organism nil)
    (setq reaction-is nil)
    (when (pathway-require-organisms *pathway*)
      (push #'reaction-organisms reaction-is)
      (setq organism (pathway-organism *pathway*)))
    (when (pathway-require-enzymes *pathway*)
      (push #'reaction-enzyme reaction-is))


    (setq successors (filter-rc-successors successors
                                  :allow    (pathway-allow *pathway*)
                                  :organism organism
                                  :reaction-is reaction-is
                                  :avoid    (pathway-avoid *pathway*)
                                  :avoid-sides t
                                  :debug debug))

    (when debug
      (format t "After filtering, ~A successors.~%"
            (length successors)))

    successors))

(defmethod organic-reactant-cost ((from compound) (r reaction) (to
compound) &key debug)
  (let (reactants products reactants2 (cost 0))
    (when debug (format t "~%organic-cost for ~A ==~A=> ~A:~% "
                  (compound-name from)
                  (reaction-id r)
                  (compound-name to)))
    (when debug (reaction-pretty-equation r))

    (when (member from (reaction-reactants r))
      (setq reactants (reaction-reactants r)))
```

```
    (when  (member from (reaction-products r))
      (setq reactants (reaction-products r)))

    (setq  reactants (remove from reactants))

    (dolist (c reactants)
      (when (compound-filter c) (push c reactants2)))

    (setq  reactants reactants2)

    (when  debug (format t " ~A~%" (mapcar #'compound-name reactants)))
    (when  reactants
      (setq cost (apply #'+ (subst 1000 0 (mapcar #'length (mapcar
#'compound-features reactants))))))
    (when  debug (format t " cost = ~A~%" cost))
    cost) )

(defmethod organic-product-cost ((from compound) (r reaction) (to
compound)  &key debug)
  (let (reactants products2 (cost 0))
    (when  debug (format t "~%organic-cost for ~A ==~A=> ~A:~% "
                   (compound-name from)
                   (reaction-id r)
                   (compound-name to)))
    (when  debug (reaction-pretty-equation r))

    (when  (member to (reaction-products r))
      (setq products (reaction-products r)))

    (when  (member to (reaction-reactants r))
      (setq products (reaction-reactants r)))

    (setq  products (remove to products))

    (dolist (c products)
      (when (compound-filter c) (push c products2)))

    (setq  products products2)

    (when  debug (format t " ~A~%" (mapcar #'compound-name products)))
    (when  products
      (setq cost (apply #'+ (subst 1000 0 (mapcar #'length (mapcar
#'compound-features products))))))
    (when  debug (format t " cost = ~A~%" cost))
    cost) )

(defmethod organism-kingdom-equal ((organism1 genome) (organism2
genome))
  (let* ( (lineage1 (genome-lineage organism1))
       (lineage2 (genome-lineage organism2))
       (kingdom1 (nth 0 lineage1))
       (kingdom2 (nth 0 lineage2)))
    (equal kingdom1 kingdom2)))

(defmethod organism-has-similar-gene-available ((organism genome)
(reaction reaction))
  (let ((organisms (remove-if-not #'genome-p (mapcar #'lookup
(reaction-organisms reaction)))))
```

```
    (dolist (o organisms)
      (when (organism-kingdom-equal organism o)
      (return-from organism-has-similar-gene-available t)))
    nil))


(defmethod edge-cost ((problem pathway-finding-problem) node action
compound)
;   (declare-ignore action)
  (let (start goal pathd
            cost
            (step-cost 1)
            organism
            (successor-cost 0)
            (carbon-cost 0)
            (organic-reactant-cost 0)
            (organic-product-cost 0)
            (xenogenic-cost 0)
            (kingdom-cost 0)
            (unannotated-cost 0)
            (organic-cost-multiplier)
            )
    (setq start (problem-initial-state problem))
    (setq goal (problem-goal problem))
    (setq pathd (compound-feature-distance start goal))

    ;; organic reactant cost
    (setq organic-cost-multiplier (+ 4 (* 10 (exp (/ pathd 20.0)))))

    (when (NOT (= (pathway-organic-reactant-cost *pathway*) 0))
      (setq organic-reactant-cost (organic-reactant-cost (node-state
node) action compound :debug nil)))

    ;; organic product cost
    (when (NOT (= (pathway-organic-product-cost *pathway*) 0))
      (setq organic-product-cost (organic-product-cost (node-state
node) action compound :debug nil)))

    ;; carbon cost
    ;;  disproportionately cost carbon distance

    (when (NOT (= (pathway-carbon-cost *pathway*) 0))
      (let ((compound-c (compound-feature compound 'c))
          (goal-c (compound-feature goal 'c)))
      (setq carbon-cost 0)
      (when (AND compound-c goal-c)
        (setq carbon-cost (abs (- compound-c
                              goal-c))))))

    ;; xenogenic cost (multiple organisms)
    (setq organism (nth 0 (pathway-organism *pathway*)))
    (when (AND organism
              (NOT (= (pathway-xenogenic-cost *pathway*) 0))
              (NOT (reaction-organism? action organism)))
      (setq xenogenic-cost 1))

    (when (AND (pathway-kingdom-cost *pathway*)
              (reaction-organisms action)
              organism)
```

```
      (unless (organism-has-similar-gene-available (lookup organism)
action)
      (setq kingdom-cost 1)))

;     (when (> kingdom-cost 0)
;       (format t "Kingdom cost.  Organism ~A, ~A, ~A~%" organism
(reaction-id action) (reaction-organisms action)))

    ;; successors cost

    (when (NOT (= (pathway-unannotated-cost *pathway*) 0))
      (setq unannotated-cost (if (reaction-genes action) 0 1)))

    (when (NOT (= (pathway-successor-cost *pathway*) 0))
      (setq successor-cost (length (successors problem compound))))

    (setq cost (+ (* (pathway-organic-reactant-cost *pathway*)
                     organic-cost-multiplier
                     organic-reactant-cost)
                  (* (pathway-unannotated-cost *pathway*)
                     unannotated-cost)
                  (* (pathway-organic-product-cost *pathway*)
                     organic-cost-multiplier
                     organic-product-cost)
                  (* (pathway-carbon-cost *pathway*)
                     carbon-cost)
                  (* (pathway-xenogenic-cost *pathway*)
                     xenogenic-cost)
                  (* (pathway-successor-cost *pathway*)
                     successor-cost)
                  (* (pathway-step-cost *pathway*)
                     step-cost)
                  (* (pathway-kingdom-cost *pathway*)
                     kingdom-cost)
                  (* (pathway-distance-cost *pathway*)
                     (compound-distance (node-state node) compound))))

;     (format t " total cost = ~A~%" cost)
    cost))

(defmethod h-cost ((problem pathway-finding-problem) state)
  (compound-distance state
                     (problem-goal problem)))

(defmethod goal-test ((problem pathway-finding-problem) node)
;  (declare-ignore node)
  (funcall (pathway-goal-test *pathway*) node *pathway*))

;;; Here we define the GENERAL-SEARCH function, and then a set of
;;; search functions that follow specific search strategies.  None of
;;; these algorithms worries about repeated states in the search.

;;; We make nodes a global parameter so that we can do multiple
searches.
(defvar *nodes* nil)

(defun print-path (node &optional (s t))
  (let (path-compounds path-reactions)
    (setq path-compounds nil)
    (setq path-reactions nil)
```

```
    (format s "~3A " (path-steps node))
    (dolist (n (solution-nodes node))
      (let ((r (node-action n))
            (c (node-state  n)))
      (pushnew r path-reactions)
      (pushnew c path-compounds)
      (if r (format s "<~11A:~8A> " (slot-value r 'enzyme) (reaction-
id r)))
      (format s "~A "
              (compound-id (node-state n)))))
    (format s "~%")))

(defun general-search (problem queuing-fn)
  "Expand nodes according to the specification of PROBLEM until we
find
  a solution or run out of nodes to expand.  The QUEUING-FN decides
which
  nodes to look at first. [p 73]"
  (let (
;     (nodes (make-initial-queue problem queuing-fn))
      node)

    (unless *nodes* (setq *nodes* (make-initial-queue problem queuing-
fn)))

    (loop (if (empty-queue? *nodes*) (RETURN nil))
        (setq node (remove-front *nodes*))
;       (when debug (print-path node))

        ;; this used to be below the goal-test
        ;; moved so that repeated bfs works better
        (funcall queuing-fn *nodes* (eliminate-reactions (expand node
problem)))
        (when debug (print-path node))
        (when (goal-test problem node)
            (format t "~6A " (length (q-elements *nodes*)))
            (RETURN node))

        )))


(defun node-g+h-cost (node)
  (+ (node-g-cost node) (node-h-cost node)))

(defun A*-no-pathmax-search (problem)
  "Search the nodes with the best f cost first.  If a node is ever
reached by
  two different paths, keep only the better path."
  (general-search problem (make-eliminating-queuing-fn #'node-g+h-
cost)))

(defun no-duplicates-breadth-first-search (problem)
  "Do breadth-first search, but eliminate all duplicate states."
  (let ((table (make-hash-table :test #'equal)))
    (general-search problem
                #'(lambda (old-q nodes)
                  (enqueue-at-end old-q (eliminate-cycles (eliminate-
all-duplicates
                                                          nodes table))))))
```

```
(defun looping-action? (node &optional (depth infinity))
  "Did this node's state appear previously in the path?"
  ;; Search up to DEPTH nodes deep in the path
  (let ((n (node-parent node)))
    (for i = 1 to depth do
       (when (null n) (return nil))
       (when (equal (node-action node) (node-action n)) (return t))
       (setf n (node-parent n)))))

(defun eliminate-reactions (nodes)
  (remove-if #'looping-action? nodes))

;; add goal-test limit capaibility to push solution to list,
;; return list on result limit
;; return list on step limit

(defun no-cycles-breadth-first-search (problem)
  "Do breadth-first search, but eliminate immediate returns to a prior
state."
  (general-search problem
              #'(lambda (old-q nodes)
                  (enqueue-at-end old-q (eliminate-cycles nodes)))))

(defstruct search-algorithm function)
(defvar A* nil)
(defvar A*-no-pathmax nil)
(setq  A* (make-search-algorithm :function #'A*-search))
(setq  A*-no-pathmax (make-search-algorithm :function #'A*-no-pathmax-
search))
(defvar Tree-A* (make-search-algorithm :function #'tree-A*-search))
;(defvar SMA (make-search-algorithm :function #'tree-SMA))
(setq BFS (make-search-algorithm :function #'no-cycles-breadth-first-
search))
;(setq BFS (make-search-algorithm :function #'no-duplicates-breadth-
first-search))
(defvar DFS (make-search-algorithm :function #'no-cycles-depth-first-
search))
(defvar *pathway* nil) ; the active pathway

(defun path-steps (node) (- (length (solution-nodes node)) 1))

(defun paths-chemically-equal (n1 n2)
  (cond ((AND (node-p n1) (node-p n2))
        (equal (mapcar '(lambda (n) (node-state n)) (solution-nodes
n1))
               (mapcar '(lambda (n) (node-state n)) (solution-nodes
n2))))
       ((AND (pathway-p n1) (pathway-p n2))
        (equal (pathway-compounds n1) (pathway-compounds n2)))))

;; return a path when current node is the "to" node
(defun goal-test-compound (node pathway)
  (equal (pathway-to pathway) (node-state node)))

(defun goal-test-compound-unique-path (node pathway)
  "true if node is a chemically unique solution from the other
results"
  (let (results)
    (when debug (format t "goal test ~A (~A)~%"
                  (compound-id (pathway-to pathway))
```

```
                (type-of node)))


    (when debug
      (when (node-action node)
      (format t " node-action: ~A~%" (reaction-id (node-action
node))))
      (format t " node-state: ~A~%"  (compound-id (node-state node))))
    (when (AND (> (path-steps node) 0)
               (equal (pathway-to pathway) (node-state node))
               (goal-test-length<= node pathway)
               (goal-test-length>  node pathway))
      (setq compounds (mapcar #'node-state (solution-nodes node)))
      (setq results (pathway-results pathway))
      (format t "~A results~%" (length results))
      (dolist (r results)
      (when (paths-chemically-equal (pathway-result-solution r) node)
        (format t "paths are chemically equal!~%")
        (print-path node)
        (print-path (pathway-result-solution r))
        (return-from goal-test-compound-unique-path nil)))
      t)))



;; return a path as long as its less than max-steps
(defun goal-test-length<= (node pathway)
  (let ((steps (path-steps node))
      (max (pathway-max-steps pathway)))
    (OR (NOT max)
      (AND (> steps 0)
           (<= steps max)))))

(defun goal-test-length> (node pathway)
  (let ((steps (path-steps node))
      (min (pathway-min-steps pathway)))
    (format t "steps = ~A, min = ~A~%" steps min)
    (OR (NOT min)
      (> steps min))) )

(defun goal-test-length= (node pathway)
  (let ((steps (path-steps node)))
  (= steps (pathway-max-steps pathway))))

;;
;; mine-pathway
;;
;; TODO:
;;   using-algorithm
;;     depth-limited-breadth-first-search
;;       modify goal test to push solutions onto a stack
;;     limit
;;   organism-exclusively? t
;;   to-less search
;;     just from and a path length
;;     goal-test is always true
;;   sort-by
;;     steps
;;     length
;;     compound-production
```

```
;;      organism
;;    with-transformation
;;      (check-box)
;;      KEGG
;;      organic
;;      inorganic
;;      cofactors
;;
;; (pathway-compare p1 p2 :graph t)
;;

(defvar *solutions* nil)
(defvar use-kegg nil)

(defun mine-pathway (&key pathway
                     from
                     to
                     in
                     knockout
                     avoid
                     allow
                     (with-algorithm A*)
                     (max-paths 1)
                     (goal-test 'goal-test-compound-unique-path)
                     min-steps
                     max-steps
                     (analyze t)
                     use-kegg
                     predict ; function to predict successors given a
compound
                     require-organisms
                     require-enzymes
                     (organic-reactant-cost 0)
                     (organic-product-cost 0)
                     (carbon-cost 0)
                     (kingdom-cost 0)
                     (successor-cost 0)
                     (xenogenic-cost 100)
                     (step-cost 0)
                     (unannotated-cost 0)
                     (distance-cost 1)
                     backwards
                     graph)
  "PathMiner"
  (check-type to pathminer::compound)
  (check-type from pathminer::compound)
;  (check-type in pathminer::genom)
;  (check-type with-algorithm pathminer::search-algorithm)
;      wrap algorithms with a simple struct
;      (defstruct search-algorithm function) (make-search-algorithm
:function #'A*-search)
;  (check-type avoid (or pathminer::compound pathminer::reaction
pathminer::enzyme boolean))
;  (check-type graph boolean)

  (let (problem result solution i (paths 0) steps initial-time)

    (cond (pathway
           (format t "Appending pathway...~%")
                                ; COPY, THEN APPEND
```

```
          (setq *pathway* (make-pathway :name (pathway-name pathway)
                                        :from (pathway-from pathway)
                                        :to (pathway-to pathway)
                                        :avoid (append avoid (pathway-avoid
pathway))
                                        :backwards backwards
                                        :min-steps (pathway-min-steps pathway)
                                        :max-steps (pathway-max-steps pathway)
                                        :goal-test (pathway-goal-test pathway)
                                        :algorithm (pathway-algorithm pathway)
                                        :organism (append in (pathway-organism
pathway))
                                        :require-organisms (pathway-require-
organisms pathway)
                                        :require-enzymes (pathway-require-
enzymes pathway)
                                        :organic-reactant-cost (pathway-
organic-reactant-cost pathway)
                                        :organic-product-cost (pathway-organic-
product-cost pathway)
                                        :carbon-cost (pathway-carbon-cost
pathway)
                                        :step-cost (pathway-step-cost pathway)
                                        :kingdom-cost (pathway-kingdom-cost
pathway)
                                        :distance-cost (pathway-distance-cost
pathway)
                                        :successor-cost (pathway-successor-cost
pathway)
                                        :unannotated-cost (pathway-unannotated-
cost pathway)
                                        :xenogenic-cost (pathway-xenogenic-cost
pathway)))
           )
          (t
           (setq *pathway* (make-pathway :from from
                                        :to to
                                        :organism in
                                        :allow (when (listp allow) (push to
allow))
                                        :backwards backwards
                                        :require-organisms require-organisms
                                        :require-enzymes require-enzymes
                                        :organic-reactant-cost organic-
reactant-cost
                                        :organic-product-cost organic-product-
cost
                                        :carbon-cost carbon-cost
                                        :successor-cost successor-cost
                                        :xenogenic-cost xenogenic-cost
                                        :unannotated-cost unannotated-cost
                                        :step-cost step-cost
                                        :distance-cost distance-cost
                                        :kingdom-cost kingdom-cost
                                        :min-steps min-steps
                                        :max-steps max-steps
                                        :goal-test goal-test
                                        :algorithm (search-algorithm-function
with-algorithm)
                                        :avoid (append avoid knockout)))))
```

```
    (multiple-value-bind
      (second minute hour date month year day-of-week dst-p tz)
      (get-decoded-time)
      (setf (pathway-name *pathway*) (format nil
"pathway_~4,'0D_~2,'0D_~2,'0D_~2,'0D_~2,'0D_~2,'0D"
                                     year
                                     month
                                     date
                                     hour
                                     minute
                                     second)))
;     (format t "Pathway ~A~%" (pathway-name *pathway*))
;     (format t " In ~A~%" (pathway-organism *pathway*))
    (when avoid
      (format t " Avoiding ")
      (dolist (a avoid)
      (format t "~A" (slot-value a 'name)))
      (format t "~%"))

;     (when allow
;       (format t " Allow ")
;       (dolist (a allow)
;         (format t "~A" (slot-value a 'name))
;       (format t "~%"))

    (setq use-kegg use-kegg)

;     (format t ":from ~A~%" (compound-id (pathway-from *pathway*)))
;     (format t ":to   ~A~%" (compound-id (pathway-to *pathway*)))
    (setq *problem* (make-pathway-finding-problem :initial-state
(pathway-from *pathway*)
                                          :goal (pathway-to *pathway*)))

    (unless (pathway-algorithm *pathway*)
      (setf (pathway-algorithm *pathway*) #'A*-search))

    ;; solutions
    (setq *nodes* nil)

    (setq initial-time (excl::get-internal-real-time))
    (loop
      (when (AND max-paths (>= paths max-paths))
      (format t "Max Paths ~A > ~A~%" paths max-paths)
      (return))
      (setq solution (funcall (pathway-algorithm *pathway*)
*problem*))
      (unless solution
      (format t "No solution.~%")
      (return-from mine-pathway NIL))
      (setq steps (path-steps solution))
      (setq result (make-pathway-result :problem  *problem*
                                :solution solution
                                :organism in))

      (when (AND max-steps (> steps max-steps))
      (format t  "Max Steps ~A > ~A~%" steps max-steps)
      (return))

      (when analyze (pathway-result-analyze result *pathway*))
```

```
      (format t "~10@A ~6@A ~10A "
            (setq delta-time (- (excl::get-internal-real-time)
initial-time))
            (setq nodes (problem-num-expanded *problem*))
            (when (> nodes 0) (/ (* delta-time 1.0) nodes)))
      (print-path solution)
      (push result (pathway-results *pathway*))
      (incf paths)
      )

    (setf (pathway-results *pathway*)
        (reverse (pathway-results *pathway*)))

    (format t "Found ~A paths.~%" paths)
;     (pathway-result-analyze result)
;     (print-chemical-solution (pathway-result-problem result)
(pathway-result-solution result))
;     (print (pathway-result-find-best-organism result))
;     (when graph (graph-pathway-result result);     result
    *pathway*
    ))

;;
;;

(defun permute-pathway (pathway &key (n 1) (ko-compounds? t) (ko-
reactions? t) (ko-enzymes? t))
  "Create add permutations to a pathway result.  Systematically
knockout compounds or reactions"
  (let ((result (car (pathway-results pathway)))
      compounds
      results)
    (dolist (result (pathway-results pathway))
      (when ko-compounds?
      (setq compounds (pathway-result-compounds result))
      (dolist (c (subseq compounds 1 (- (length compounds) 1)))
        (setq p2 (mine-pathway :pathway pathway :avoid (list c)))
        (when p2 (push (car (pathway-results p2)) results))))

      (when ko-reactions?
      (setq reactions (pathway-result-reactions result))
      (dolist (r (subseq reactions 1 (- (length reactions) 1)))
        (setq p2 (mine-pathway :pathway pathway :avoid (list r)))
        (when p2 (push (car (pathway-results p2)) results))))

      (when ko-enzymes?
      (setq enzymes (mapcar #'reaction-enzyme (pathway-result-
reactions result)))
      (format t "Enzymes: ~A~%" enzymes)
      (dolist (e (subseq enzymes 1 (- (length enzymes) 1)))
        (format t "avoiding ~A~%" (mapcar #'enzyme-ec e))
        (setq p2 (mine-pathway :pathway pathway :avoid e))
        (when p2 (push (car (pathway-results p2)) results)))))

      (setf (pathway-results pathway) (append (pathway-results
pathway) results))
      pathway
      ))

;;
```

```
(defmethod pathway-result-find-best-organisms ((p pathway-result) &key
print?)
  (let (organisms unique best)
    (dolist (reaction (pathway-result-reactions p))
      (unless (reaction-p reaction) (break))
      (dolist (o (reaction-organisms reaction))
      (push o organisms)
      (pushnew o unique)))
    (dolist (organism unique)
      (push (cons organism (count organism organisms)) best))
    (setq best (sort best #'(lambda (x y) (> (cdr x) (cdr y)))))
    (when print?
      (dolist (b best)
      (when (boundp (car b))
      (setq organism (symbol-value (car b)))
      (when (genome-p organism)
        (format t "~A ~A~%" (cdr b) (genome-name organism))))))
    best))


(defmethod pathway-find-best-hybrid ((p pathway) &key olist organisms
(num-organisms 2) (max-paths 6))
  (let (result
      critical
      pathway
      tmp-olist
      olists
      (paths 0))

    (print "pathway-find-best-hybrid")
    (setq result (car (pathway-results p)))
 ;   (unless result (return))
    ;; really want to find the critical enzymes,
    ;; but for now, least organisms somewhat implies critical enzymes

    (unless organisms
      (setq organisms (mapcar #'car (feature-difference (pathway-
result-organisms result) olist)))
      (setq critical (utilities::flatten (mapcar #'enzyme-organisms
(pathway-result-critical result))))
      (setq organisms (utilities::flatten (append critical (feature-
difference organisms critical))))
      (setq organisms (adjoin 'eco organisms))
      (print organisms)
      )

    (when (= (length olist) (- num-organisms 1))
      (dolist (o organisms)
      (unless (member o olist)
        (setq tmp-olist (append (list o) olist))
        (when (setq pathway (mine-pathway :to (pathway-to p)
                                  :from (pathway-from p)
                                  :in tmp-olist
                                  :analyze nil))
          (format t "~A~%" tmp-olist)
          (pushnew tmp-olist olists)
          (incf paths)
          (when (> paths max-paths)
```

```
            (return-from pathway-find-best-hybrid olists))))))

    (when (= (length olist) (- num-organisms 1))
      (return-from pathway-find-best-hybrid NIL))

    (dolist (o organisms)
      (unless (member o olist)
      (setq tmp-olist (append (list o) olist))
      (when (pathway-find-best-hybrid p :olist tmp-olist :organisms
organisms)
        (return-from pathway-find-best-hybrid tmp-olist))))
    ))

(defmethod pathway-permute ((p pathway))
  "Systematically knockout steps in the pathway"
  )

(defun fb (b d N)
  (- (/ (- (expt b (+ d 1)) 1) (- b 1)) N))

(defun b* (N d)
  (newton:newton #'(lambda (b) (fb b d N)) 2)
  )

(defmethod pathway-result-analyze ((p pathway-result) (pathway
pathway))
  (let (compound reaction clist solution-node)

    ;; compute statistics for solution-node
    (setq solution-node (pathway-result-solution p))
    (setf (pathway-result-d p) (node-depth solution-node))
    (setf (pathway-result-N p) (problem-num-expanded (pathway-result-
problem p)))
    (setf (pathway-result-f p) (node-f-cost solution-node))
    (setf (pathway-result-b* p) (b* (pathway-result-N p)
                                    (pathway-result-d p)))


    ;; make clist
    (dolist (n (solution-nodes (pathway-result-solution p)))
       (push (node-state  n) clist))

    (dolist (n (solution-nodes (pathway-result-solution p)))
       (setq compound (node-state  n))
;      (print (compound-name compound))
       (when (reaction-p (node-action n))
         (setq reaction (node-action n))
;           (print reaction)
;           (print (reaction-id reaction))
         (push (reaction-id reaction) (slot-value p 'reactions))
;        (push (cons compound reaction) (pathway-result-
;        (format t "~a~%" (reaction-id reaction))
         (when (member compound (reaction-reactants reaction))
           (dolist (c (reaction-reactants reaction))
             (unless (member c clist)
;            (format t " > ~A~%" (compound-name c))
             (push c (slot-value p 'products))))
           (dolist (c (reaction-products reaction))
             (unless (member c clist)
;            (format t " < ~A~%" (compound-name c))
```

```
            (push c (slot-value p 'reactants)))))

          (when (member compound (reaction-products reaction))
            (dolist (c (reaction-reactants reaction))
              (unless (member c clist)
;               (format t " < ~A~%" (compound-name c))
                (push c (slot-value p 'reactants))))
            (dolist (c (reaction-products reaction))
              (unless (member c clist)
;               (format t " > ~A~%" (compound-name c))
                (push c (slot-value p 'products) ))))
          )

;       (format t "~a~%" (compound-id compound))
        (push (compound-id compound) (slot-value p 'compounds))
        )
    (setf (slot-value p 'compounds) (reverse  (slot-value p
'compounds)))
    (setf (slot-value p 'reactions) (reverse  (slot-value p
'reactions)))
    (setf (slot-value p 'products)  (reverse  (slot-value p
'products)))
    (setf (slot-value p 'reactants) (reverse  (slot-value p
'reactants)))
    (setf (slot-value p 'organisms) (pathway-result-find-best-
organisms p))

    (dolist (c (subseq (slot-value p 'compounds) 0 (length (slot-value
p 'compounds)))) (pushnew c (pathway-compounds pathway)))
    (dolist (r (subseq (slot-value p 'reactions) 0 (length (slot-value
p 'reactions)))) (pushnew r (pathway-reactions pathway)))
;    (setf (slot-value p 'critical)  (pathway-result-find-critical-
enzymes p))
    )
  p)

(defmethod print-object ((p pathway) stream)
  (dolist (pr (pathway-results p))
    (format stream "~A~%" pr)))

(defmethod print-object ((pr pathway-result) stream)
  (print-chemical-solution (pathway-result-problem pr) (pathway-
result-solution pr) :stream stream)
  (format stream "Net:~%")
  (format stream "~A" (mapcar #'compound-name  (pathway-result-
reactants pr)))
  (format stream "~A" (mapcar #'compound-name  (pathway-result-products
pr))))

;(defmethod print-object ((p pathway) stream)
;  (dolist (r (pathway-results p))
;;    (print-chemical-solution (pathway-result-problem r) (pathway-
result-solution r))
;    (format t "~A~%~A~%~A~%" r (pathway-result-find-best-organisms r)
(reverse (pathway-result-find-best-organisms r))))
;  )

(defun pathway-find-critical-enzymes (p)
  (let (result)
    (setq result (car (pathway-results p)))
```

```
    (pathway-result-find-critical-enzymes result)))

(defmethod pathway-result-find-critical-enzymes ((r pathway-result))
  (print "find critical enzymes")
  (let (reactions enzymes critical)
    (setq reactions (pathway-result-reactions r))
    (setq enzymes (mapcar #'reaction-enzyme reactions))
    (dolist (e enzymes)
      (unless (mine-pathway :pathway pathway :avoid (list e) :analyze
nil)
      (format t "Knocking out ~A"  (enzyme-ec e))
      (push e critical)))
    (setf (pathway-result-critical r) critical)))


(defun pathway-transgenic-analysis (pathway)
  (let ((result (car (pathway-results pathway)))
      (organism (pathway-organism pathway))
      exozymes)
  (format t "  ~A~%" (genome-name (symbol-value (car organism))))
  (dolist (r (pathway-result-reactions result))
    (when (reaction-p r)
      (unless (reaction-organism? r (car organism))
      (dolist (e (reaction-enzyme r))
        (when (enzyme-p e)
          (pushnew e exozymes)
          (format t "~%+ ~A ~A~%" (enzyme-ec e) (enzyme-name e))
          (setq genes (slot-value e 'genes))
          (dolist (g genes)
            (when (boundp (car g))
            (setq genome (symbol-value (car g)))
            (format t "  ~20A ~A~%" (genome-name genome) (cadr g))))
          )))))
  (reverse exozymes)))


(defun print-chemical-solution (problem node &key (stream t))
  "Print a table of the actions and states leading up to a
iosolution."
  (let (enz-ec
      enz-name enz-orgs enz-org-genes
      reaction-organism-genes
      *print-pretty*
      reaction r-name ec r-maps compound c-name rr rp g h f preferred
available reactant product product-id reactant-id d N (actual-distance
0) cofactor)
    (if node
      (progn

            (format stream "~10A ~60A ~20A ~15A ~15A ~2@A,~2A ~10@A =
~10@A + ~4@A ~70A ~70A ~50A ~2A ~70A ~S~%"
                  "Reaction"
                  "Compound"
                  "Formula"
                  "E.C."
                  "EndoGenes"
                  ""
                  ""
                  "f"
                  "g"
```

```
                    "h"
                    "Enzyme Name"
                    "Reactants"
                    "Cofactor"
                    "    "
                    "Products"
                    "Organisms"
                    )
        (for each n in (solution-nodes node) do
             (setq r-name "")
             (setq rr-names "")
             (setq rp-names "")
             (setq r-direction "")
             (setq ec "")
             (setq compound (node-state  n))
             (if (reaction-p (node-action n))
               (progn
                 (setq g (node-g-cost n))
                 (setq h (node-h-cost n))
                 (setq f (node-f-cost n))
                 (setq d (node-depth n))
                 (setq reaction (node-action n))
           (setq r-name (reaction-id reaction))
                 (setq rr-names (mapcar #'compound-name (reaction-
reactants reaction)))
                 (setq r-direction (reaction-direction reaction))
                 (setq rp-names (mapcar #'compound-name (reaction-
products reaction)))
                 (setq.ec (reaction-enzyme reaction))

                 (setq enz-name nil enz-ec nil enz-orgs nil)

                 (setq organism (car (pathway-organism *pathway*)))

                 (setq reaction-organism-genes (reaction-organism?
reaction organism))
                 ;;
                 ;; Fix this.
                 ;; We kludged in multiple enzymes, so this needs to
be fixed.
                 ;;
                 (when (listp ec) (setq ec (nth 0 ec)))
                 (when (enzyme-p ec)
                   (setq ligand-enzyme (symbol-value (read-from-string
(format nil "ligand::~A" (enzyme-ec ec)))))
                   (setq cofactor (ligand::enzyme-cofactor ligand-
enzyme))
                   (setq enz-name (enzyme-name ec))
                   (setq enz-ec (enzyme-ec ec))
                   (setq enz-orgs (enzyme-organisms ec)) ; change for
multiple enzymes
                   (setq genes (reaction-genes reaction)))
;                     (setq r-maps (kegg-map-name (symbol-value
(reaction-maps reaction))))
;                     (setq rr (reaction-main-reactant reaction))
;                     (setq rp (reaction-main-product  reaction))
;                     (cond ((eql compound rr) (setq reactant rp)
(setq product rr))
;                           ((eql compound rp) (setq reactant rr)
(setq product rp)))
```

```
;                       (setq available (AND (available? reaction)))
;                       (setq preferred (OR (prefer? reaction) (prefer?
product)))
;                       (setq product-id (compound-id product))
;                       (setq reactant-id (compound-id reactant))))


                    ))
                (when (AND product reactant) (incf actual-distance
(compound-distance product reactant)))

                (setq c-name (format nil "~A ~A" (compound-id compound)
(compound-shortest-name compound)))
                (format stream "~10A ~60A ~20A ~15A ~15A ~2@A,~2A ~10@A =
~10@A + ~4@A ~70A ~70A ~50A ~2A ~70A ~1000A"
                        r-name
                        c-name
                        (ligand::compound-formula (compound-ligand
compound))
                        enz-ec
                        (first reaction-organism-genes)
                        (length reaction-organism-genes)
                        (length genes)
                        f
                        g
                        h

                        enz-name
                        ;(length enz-orgs)
                        rr-names
                        cofactor
                        r-direction
                        rp-names
                        enz-orgs
                        )

;                   (when (AND enz-orgs (listp enz-orgs))
;                     (setq enz-orgs (reverse (sort enz-orgs #'string-
greaterp)))
;                     (dolist (o enz-orgs) (format t "~a " o)))

                (format stream "~%")
                )
                (format stream "~%~%Total of ~D node~:P expanded. Cost
~D. Length ~D. b* ~D~%~%"
                 (problem-num-expanded problem) (node-f-cost node)
                 d
                 (b* (problem-num-expanded problem) d))
          )

        (format stream "~&No solution found.~&"))

      ))

;
;(defun print-chemical-solution (problem node)
;  "Print a table of the actions and states leading up to a
iosolution."
```

```
;  (let (enz-ec enz-name enz-orgs reaction r-name ec r-maps compound
c-name rr rp g h f preferred available reactant product product-id
reactant-id d N (actual-distance 0) cofactor)
;    (if node
;        (progn
;          (format t "~&Action ~10T State~%====== ~10T =====~%")
;          (for each n in (solution-nodes node) do
;               (setq r-name "")
;               (setq rr-names "")
;               (setq rp-names "")
;               (setq r-direction "")
;               (setq ec "")
;               (setq compound (node-state  n))
;               (if (reaction-p (node-action n))
;                   (progn
;                     (setq g (node-g-cost n))
;                     (setq h (node-h-cost n))
;                     (setq f (node-f-cost n))
;                     (setq d (node-depth n))
;                     (setq reaction (node-action n))
;                     (setq r-name (reaction-id reaction))
;                     (setq rr-names (mapcar #'compound-name
(reaction-reactants reaction)))
;                     (setq r-direction (reaction-direction reaction))
;                     (setq rp-names (mapcar #'compound-name
(reaction-products reaction)))
;                     (setq enzymes (reaction-enzyme reaction))
;;                     (setq ligand-enzyme (symbol-value (read-from-
string (format nil "~A" (enzyme-ec ec)))))
;;                   (setq cofactor (ligand::enzyme-cofactor ligand-
enzyme))
;
;                     (setq enz-name nil enz-ec nil enz-orgs nil)
;
;                     (setq orgs (reaction-organisms reaction))
;
;                     (setq enz-name (enzyme-name enzymes))
;                     (setq enz-ec (enzyme-ec enzymes))
;                     (setq enz-orgs (enzyme-organisms enzymes))
;;                     (setq r-maps (kegg-map-name (symbol-value
(reaction-maps reaction))))
;;                     (setq rr (reaction-main-reactant reaction))
;;                     (setq rp (reaction-main-product  reaction))
;;                     (cond ((eql compound rr) (setq reactant rp)
(setq product rr))
;;                           ((eql compound rp) (setq reactant rr)
(setq product rp)))
;
;;                     (setq available (AND (available? reaction)))
;;                     (setq preferred (OR (prefer? reaction) (prefer?
product)))
;;                     (setq product-id (compound-id product))
;;                     (setq reactant-id (compound-id reactant))))
;                   ))
;                (when (AND product reactant) (incf actual-distance
(compound-distance product reactant)))
;
;               (setq c-name (format nil "~A ~A" (compound-id
compound) (compound-name compound)))
```

```
;                (format t "~6A ~10T ~50A ~20A ~45A ~70A ~2A=~2.A+~2A
~2A ~70A ~50A ~2A ~70A"
;                        r-name
;                        c-name
;                        (ligand::compound-formula (compound-li gand
compound))
;                        enz-ec
;                        (CAR enz-name)
;                        f
;                        g
;                        h
;                        (length enz-orgs)
;                        rr-names
;                        cofactor
;                        r-direction
;                        rp-names
;                        )
;                (setq orgs (reverse (sort orgs #'string-greaterp)))
;                (dolist (o orgs) (format t "~a " o))
;                (format t "~%")
;                )
;                (format t "====== ~10T =====~%Total of ~D node~:P
expanded. Cost ~D. Length ~D. b* ~D~%~%"
;                    (problem-num-expanded problem) (node-f-cost  node)
;                    d
;                    (b* (problem-num-expanded problem) d))
;          )
;
;       (format t "~&No solution found.~&"))
;
;    ))


(defun compare-searches (&key (endpoints (list (list alpha-d-glucose
pyruvate)
                                               (list pyruvate acetate)
                                               (list pyruvate l-alanine)
                                               (list pyruvate succinate)
                                               (list prpp imp)))
                              (algorithms (list BFS DFS A*)))
  (let (answer)
  (dolist (endpoint endpoints)
    (setq from (nth 0 endpoint))
    (setq to (nth 1 endpoint))
    (dolist (algorithm algorithms)
      (format t "(mine-pathway :from ~A :to ~A :with-algorithm  ~A)~%"
            (compound-name from)
            (compound-name to)
            (search-algorithm-function algorithm))
      (setq pathway (mine-pathway :from from :to to :with-algorithm
algorithm))
      (setq result (car (pathway-results pathway)))
      (setq N (pathway-result-N result))
      (setq d (pathway-result-d result))
      (setq b* (pathway-result-b* result))
      (setq f (pathway-result-f result))
      (push (format nil "~50A ~50A ~100A N ~3A d ~3A F ~3A b* ~3A~%"
                (compound-name from)
                (compound-name to)
                (search-algorithm-function algorithm)
```

```
                N d f b*) answer)
    ))
  answer))

;(defmethod graph-pathway-result ((p pathway-result))
;  (when (not e3d::epiphany-stream)
;    (e3d::epiphany))
;  (epiphany-pathway-layout-send (pathway-result-compounds p)))



;; Should go to math.newton
;;


;(defun pathway-tex (clist)
;  (let (figfile (latex-string "") table row reactant product deltas)
;    (setq latex-string (format nil "~a" (latex::center)))
;    (push (list "Substrate" "-" "+" "Product") table)
;    (dolist (c clist)
;      (setq reactant product)
;      (setq product c)
;      (when (AND reactant product)
;        (setq deltas (compound-fvect-difference product reactant))
;        (setq productfigfile  (format nil
"/home/dan/research/doc/project/pathminer/figs/ligand-eps/~a.gif.eps"
(car (compound-entry product))))
;        (setq row (list (compound-latex reactant)
;                        (latex::tiny (fvect->flist (nth 2 deltas)))
;                        (latex::tiny (fvect->flist (nth 1 deltas)))
;                        (compound-latex product)))
;        (push row table))
;      )
;    (setq table (reverse table))
;    (setq latex-string (format nil "~a~a" latex-string (latex::table
table :alignment "p{3cm}" :separator "|" :hline t)))
;
;    latex-string))


;;
;; Reference Pathways
;;

(defun setup-pathways ()
  (setq mendz-glycolysis (list alpha-d-glucose
                           alpha-d-glucose-6-phosphate
                           beta-d-fructose-6-phosphate
                           beta-d-fructose-1-6-bisphosphate
                           dihydroxyacetone-phosphate
                           d-glyceraldehyde-3-phosphate
;                        glyceraldehyde-3-phosphate
                           3-Phospho-D-glyceroyl-phosphate
                           3-phospho-d-glycerate
                           2-phospho-d-glycerate
                           phosphoenolpyruvate
                           pyruvate))

  (setq tca-compounds (list citrate
```

```
                        cis-aconitate
                        isocitrate
                        alpha-ketoglutaric-acid
                        succinate
                        fumarate
                        s-malate
                        oxaloacetate)))


(defmethod pathway-equations ((pathway pathway))
  (let (matrix matrices equations)
    (format t "~%")
    (dolist (result (pathway-results pathway))
      (dolist (r (pathway-result-reactions result))
      (format t "~A ~A~%" (reaction-id r) (reaction-pretty-equation
r)))
      (setq matrix (pathway-result-equation result))
      (push matrix matrices)
      (print-matrix matrix)
      (push (matrix-equation matrix) equations))
    equations))

(defmethod pathway-result-equation ((result pathway-result))
  (let ((cindex 0) reactions compounds equation reactants products
compound matrix)

    (setq reactions (pathway-result-reactions result))

    (dolist (r reactions)
      (dolist (c (reaction-reactants r))
      (pushnew (compound-id c) compounds))
      (dolist (c (reaction-products r))
      (pushnew (compound-id c) compounds)))
    (setq compounds (reverse compounds))

    (dolist (r reactions)
      (setq compound (nth cindex (pathway-result-compounds result)))
      (incf cindex)
      (setq array (make-array (length compounds) :initial-element 0))
      (setq equation (reaction-equation r))

      (when (member compound (reaction-reactants r))
      (setq reactants (nth 0 equation))
      (setq products  (nth 2 equation)))

      (when (member compound (reaction-products r))
      (setq reactants (nth 2 equation))
      (setq products  (nth 0 equation)))

      (dolist (c reactants)
      (setf (aref array (position (car c) compounds)) (- (cadr c))))

      (dolist (c products)
      (setf (aref array (position (car c) compounds)) (cadr c)))

      (push array matrix))

    (setq matrix (reverse matrix))

    (push compounds matrix)
```

```
    (matrix-equation matrix)))

(defun print-matrix (matrix)
  (format t "~%")
  (dolist (l (car matrix))
    (format t "~6@A " (compound-abbreviate (symbol-value l))))
  (format t "~%")
  (dolist (a (cdr matrix))
    (dotimes (i (length (car matrix)))
      (format t "~6@A " (aref a i)))
    (format t "~%")))

(defun matrix-equation (matrix)
  (let (first
      compounds
      array
      string
      (equation (make-string-output-stream)))
    (setq array (make-array (length (car matrix)) :initial-element 0))
    (setq compounds (car matrix))
    (dolist (a (cdr matrix))
      (dotimes (i (length array))
      (incf (aref array i) (aref a i))))

    (setq first t)
    (dotimes (i (length array))
      (when (< (aref array i) 0)
      (unless first
        (format equation" + "))
      (setq first nil)
      (format equation"~A ~A"
            (if (NOT (= -1 (aref array i) )) (- (aref array i)) "")
            (compound-name (symbol-value (nth i compounds))))))
    (format equation" => ")

    (setq first t)
    (dotimes (i (length array))
      (when (> (aref array i) 0)
      (unless first
        (format equation" + "))
      (setq first nil)
      (format equation "~A ~A"
            (if (NOT (= 1 (aref array i) )) (aref array i) "")
            (compound-name (symbol-value (nth i compounds))))))
    (setq string (get-output-stream-string equation))
    string))


(defun pathway-result-successor-table (pathway-result &optional
organism)
  (let (compounds rc-successors)
    (setq compounds (pathway-result-compounds pathway-result))
    (dolist (compound compounds)
      (format t "~A~%" (compound-name compound))
      (setq successors (compound-rc-successors compound))
      (setq rc-successors (filter-rc-successors successors :organism
organism))
      (dolist (successor rc-successors)
      (pushnew successor successors))
```

```
      (dolist (successor rc-successors)
      (setq enzymes (slot-value (car successor) 'enzyme))

      (format t " = ~50A ~50A => ~A~%"
            enzymes
            (reaction-organism? (car successor) (car organism))
            (compound-name  (cdr successor))
            )))))

(defun pathway-successor-table (pathway &optional organism)
  (pathway-result-successor-table (car (pathway-results pathway))
organism))

(defun pathway-description (pathway)
  (let ((stream (make-string-output-stream)))
    (format stream "From ~A To ~A" (compound-name (pathway-from
pathway)) (compound-name (pathway-to pathway)))
    (when (pathway-organism pathway)
      (format stream " in ~A" (genome-name (pathway-organism
pathway))))
    (setq string (get-output-stream-string stream))
    ))


(defun pathway-table (pathway &key lyx)
  (let (previous
      all-compounds
      all-transformations
      sorted-compounds)
    (dolist (result (pathway-results pathway))
      (let ((node (pathway-result-solution result)))
      (for each n in (solution-nodes node) do
           (let* ((compound (node-state  n))
                (reaction (node-action n)))
             (pushnew compound all-compounds)
             (format t "~A (~A)~%" (compound-id compound) (when
previous (compound-id previous)))
             (when reaction
             (let* ((transformations (reaction-lkup-transformations
reaction)))
               (dolist (transformation transformations)
                 (let* ((transformation (lookup transformation))
                      (enzyme (car (transformation-enzymes
transformation)))
                      (ec (when (enzyme-p enzyme) (enzyme-ec enzyme)))
                      (ename (when (enzyme-p enzyme) (enzyme-name
enzyme))))
                   (let ((reactant (lookup (transformation-reactant
transformation)))
                       (product  (lookup (transformation-product
transformation))))
                   (when (AND (equalp reactant previous)
                            (equalp product  compound))

                     (format t " ~20A ~A: " ec ename)
                     (format t " ~A: " (transformation-id
transformation))
                     (format t "  ~A=>~A~%" (compound-id reactant)
(compound-id product))
                     (pushnew transformation all-transformations))
```

```
                    )))))
              (setq previous compound)))
      (format t "~%~%---~%~%")))

    (format t "~A compounds~%" (length all-compounds))
    (format t "~A transformations~%" (length all-transformations))

    (setq all-compounds (reverse all-compounds))
    (dolist (compound all-compounds)
      (let ((name (compound-shortest-name compound)))
      (format t "~A~%" name)

      (dolist (transformation all-transformations)
        (let ((reactant (lookup (transformation-reactant
transformation)))
              (product  (lookup (transformation-product
transformation)))
              (id (transformation-id transformation)))
          (when (equalp reactant compound)
            (format t "  ~A: ~A=>~A~%" id (compound-id reactant)
(compound-id product)))))))))

    ))
```